Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ
КАФЕДРА «ПРОМИСЛОВОЇ ЕЛЕКТРОНІКИ»

СУЧАСНІ НАПРЯМКИ КОМП'ЮТЕРНОЇ І МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ.

РОЗДІЛ 3. АРХІТЕКТУРА СУЧАСНИХ МІКРОКОНТРОЛЕРІВ

КОНСПЕКТ ЛЕКЦІЙ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для аспірантів,
які навчаються за спеціальністю 171 «Електроніка»,
спеціалізацією «Електронні компоненти і системи»

Київ 2019

Сучасні напрямки комп'ютерної і мікропроцесорної техніки. Розділ 3. Архітектура сучасних мікроконтролерів: конспект лекцій [Електронний ресурс]: .для студентів підготовки в галузі знань 17 Електроніка та телекомунікація за спеціальністю 171 Електроніка за спеціалізацією «Електронні системи» / КПІ ім. Ігоря Сікорського ; уклад.: Т. О. Терещенко, Ю.С. Ямненко, Ю.В.Хохлов – Електронні текстові данні (1 файл: 8525 кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2019. – 204 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 7 від 01.04.2019 р.) за поданням Вченої ради факультету електроніки (протокол №03/2019 від 25.03.2019 р.)

Електронне мережне навчальне видання

СУЧАСНІ НАПРЯМКИ КОМП'ЮТЕРНОЇ І МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

Конспект лекцій для студентів напрямку підготовки 171 Електроніка

Укладачі: *Терещенко Тетяна Олександрівна,* докт. техн. наук
Ямненко Юлія Сергіївна, докт. техн. наук
Хохлов Юрій Вітальович, канд.техн. наук

Відповідальний редактор *Миколаєць Д.А.* доцент кафедри промислової електроніки, канд. техн. наук, проф.

Рецензенти: *Михайлов С.Р.,* доцент кафедри електронних приладів та пристроїв, канд. техн. наук, доц.

Навчальний посібник «Сучасні напрямки комп'ютерної і мікропроцесорної техніки: конспект лекцій.для студентів напрямку підготовки 171» сприяє набуттю практичних навичок створення розробки апаратної частини та програмного забезпечення мікропроцесорних систем на базі ARM процесорів та методів їх налагодження.

Київ – 2019

Зміст

Розділ 3. Архітектура сучасних мікроконтролерів

Тема 3.1. ARM процесори

1. Історичні аспекти розвитку ARMпроцесорів

Процесори ARM - нинішній безумовний лідер мікропроцесорного ринку мобільних і інтегрованих рішень. На рис.1.1 зображено узагальнений шлях розвитку мобільних рішень (апаратів стільникового зв'язку, кишенькових обчислювальних пристроїв, тощо). Поряд із самим розвитком технічної бази мобільних рішень проілюстровано розвиток безпосередньо поколінь зв'язку – 2G -> 3G ->LTE.

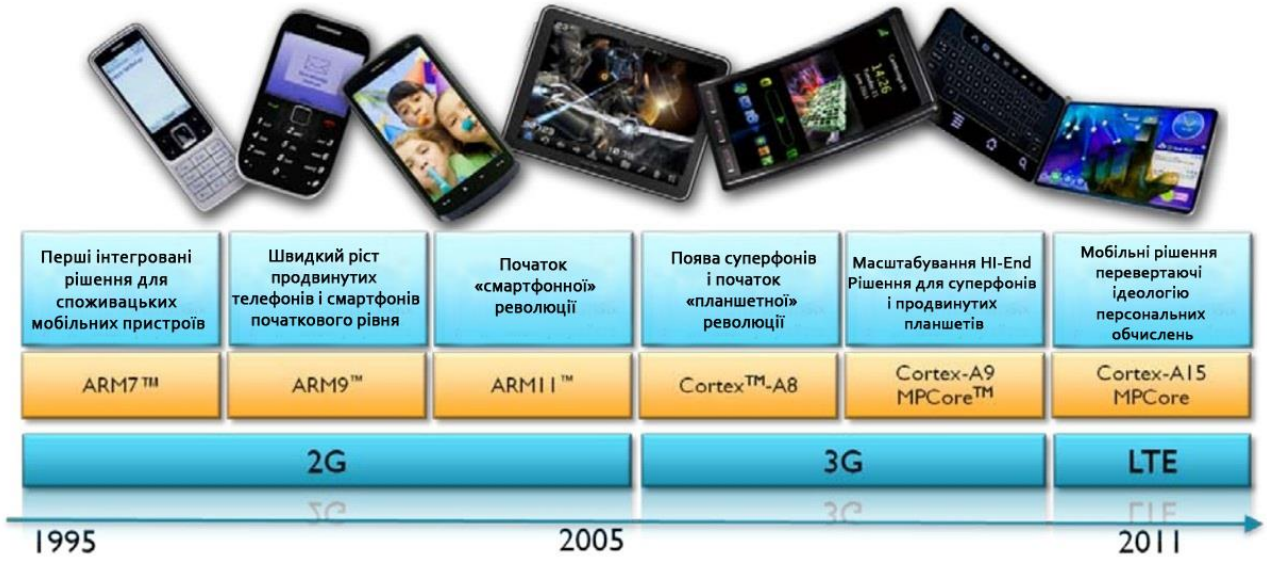


Рис. 1.1. Еволюція мобільних рішень.

Саме необхідність в процесорі підвищеної потужності, здатному працювати з графічним інтерфейсом користувача, призвела на початку 80-х років минулого сторіччя британську компанію AcornComputersдо думки про необхідність відмови від готових, але малопотужних рішень, пропонованих партнерами MOS Technology і Motorola, і запуску розробки нового власного процесора.

Учасники проекту BBC Micro створили для Acorn мікропроцесор на архітектури RISC (Restricted (reduced) instructionsetcomputer - комп'ютер з скороченим набором команд), яка є альтернативою популярної архітектури CISC

(Complexinstructionsetcomputer). RISC-архітектура пропонувала оптимізацію обчислювального процесу за рахунок реалізації складних функцій не за допомогою єдиної комплексної команди, як це робилося в CISC, а за допомогою набору більш простих команд. При такому підході арифметико-логічний пристрій істотно спрощується, що дозволяє додати в схему процесора більшукількість регістрів. Збільшення кількості регістрів знижує необхідність частого звернення до повільною оперативної пам'яті.

Вперше 32-розрядна архітектура ARM першої версії (ARMv1) була застосовано в процесорі ARM1 і використовувалася як заміна малопотужним CISC-процесорам в комп'ютері BBC Micro. Зовнішній вигляд цього процесору зображена на рис. 1.2.

Рис.1.2. Перший процесор ARM1 для Acorn зробила компанія-партнер VLSI

У фірмі ARM обрали вектор розвитку мікропроцесорів спрямований у бік технологій ASIC і ASSP.

Технологія ASSP (Application-specificstandardproducts) передбачає розробку простих, але в той же час універсальних по застосуванню компонентів - наприклад, апаратних декодерів звуку і відео.

Технологія ASIC (Application-specificintegratedcircuit) на противагу ASSP передбачає створення інтегральних мікросхем, що спеціалізуються на вирішенні деякого обмеженого кола завдань. До ASIC-рішень можна віднести роутери, мобільні телефони й ігрові консолі.

В основі ASIC-системи лежить процесорне ядро, контролери пам'яті і периферійних пристроїв.

Отже, завдяки такій самодостатності та поєднанню технологій ASIC та ASSP можна за дуже короткий термін можна створити новий пристрій довільної конфігурації, адаптований саме для використання під конкретно визначені задачі.

2. Основні характеристики ядра ARM7

Нижче наведені основні якісні та кількісні характеристики мікропроцесорного ядра ARM 7:

• 32-розрядний RISC процесор (32-розрядні шини даних і адреси) з продуктивністю 17 MIPS при тактовій частоті 25 МГц (пікова продуктивність 25 MIPS)

• 32-розрядна адресація - лінійний адресний простір в 4 Гбайта - виключає необхідність сегментованої, розділеної на банки або оверлейної пам'яті

• Тридцять один 32-розрядний регістр загального призначення і шість регістрів стану

• Регістри адрес, запису і конвеєру

• Циклічний пристрій зсуву і перемножувач

• Трирівневий конвеєр (вибірка команди, її декодування і виконання)

• Робочі режими BigEndian і LittleEndian

• Напруга живлення 3,3 і 5 В

• Мале споживання 0,6 мА / МГц, при виготовленні по CMOS технології з топологічними нормами 0,8 мкм.

• Повністю статична робота, що дозволяє додатково знижувати споживання за рахунок зменшення тактової частоти, що є ідеальним для критичних до споживання додатків

• Швидкий відгук на переривання додатків реального масштабу часу

• Підтримка систем віртуальної пам'яті

• Проста але потужна система команд

Структурна схема ядра ARM 7 зображена на рис.1.3

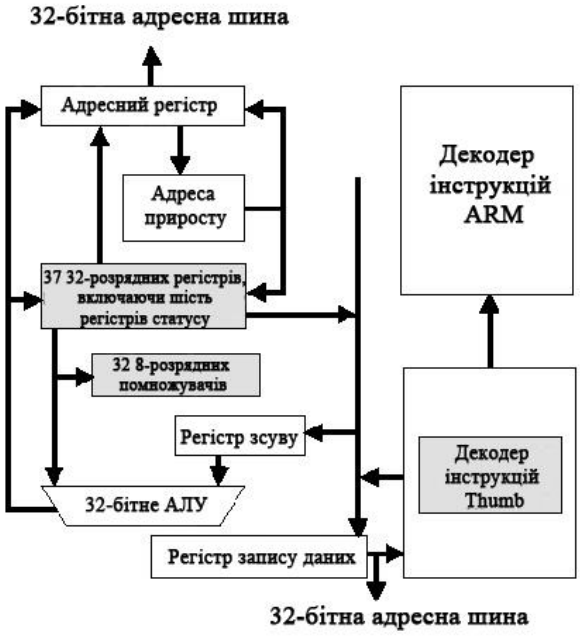


Рис.1.3. Структурна схема ядра ARM7

Особливістю структурної схеми ядра ARM7 є наявність декодеру інструкцій Thumbта декодеру інструкцій ARM. За допомогою їх використання вдається компресувати 32-розрядні команди у 16-розрядні, а потім безпосередньо перед їх виконанням декомпресувати. Далі технологія Thumbбуде детально пояснена. За допомогою спрощеного арифметико-логічного пристрою у наявності є 37 32-розрядних регістри, включаючи шість регістрів статусу

RISC процесор ARM7 CPU містить тридцять один 32-розрядних регістрів загального призначення й шість регістрів стану, циклічний пристрій зсуву і блок множення, трирівневий конвеєр, який суміщує у часі цикли вибірки команди, її декодування й виконання.

32-розрядна система команд ядра ARM7 містить одинадцять базових типів команд:

• Два типи використовують вбудований арифметико-логічний пристрій, циклічний пристрій зсуву і перемножувач при операціях над даними в банку з 31 регістра, форматом по 32 розряду кожен;

• Три класи команд управління переміщенням даних між пам'яттю і регістрами, один оптимізований на забезпечення гнучкості адресації, інший під швидке контекстне перемикання і третій під підкачку даних;

• Три команди управляють потоком і рівнем привілею виконання;

• Три типи призначені для управління зовнішніми співпроцесорами, що дозволяє розширити функціональні можливості системи команд за межами ядра.

Система команд ARM добре обробляється компіляторами мов високого рівня. На відміну від деяких RISC процесорів, процесор ARM7, при виникненні необхідності в деякому зменшенні обсягу кодів, допускає програмування і на асемблері.

Надаючи на ліцензійній основі ядро ARM7 своїм кремнієвим партнерам фірма ARM на його основі розробила мікроконтролери ARM7100, ARM7500 і ARM7500FE.

Опис мікроконтролеру ARM7100 наведений нижче з метою кращого розкриття можливостей ядра ARM7.

Мікроконтролер ARM7100 можна назвати мікроконтролером широкого застосування, оскільки він орієнтований на використання в таких пристроях як: персональні інформаційні пристрої (PDA) та органайзери, інтелектуальні мобільні телефони і багатофункціональні пейджери, кишенькові вимірювальні

пристрої та системи збору даних. Мікроконтролер організований за модульним принципом з використанням внутрішньої шини AMBA, що організовує взаємодію ядра із стандартними бібліотечними осередками периферії.

Необхідно відзначити, що переведення ядра на технологію зі зменшеними топологічними нормами дозволяє як підвищити його продуктивність, так і ще більше знизити споживання.

Основні характеристики ARM7100:

• Продуктивність 18,4 MIPS при тактовій частоті 18,4 МГц і напрузі живлення 3,3 В

• Споживання 66 мВт при напрузі живлення 3,3 В

• Вбудований єдиний кеш команд і даних ємністю 8 Кбайт

• Інтерфейс ROM і розширення (сегменти 8x256 Мбайт 8 -, 16 - і 32-розрядні)

• Контролер DRAM з підтримкою швидкого сторінкового режиму (8 -, 16 - і 32-розрядних)

• адресний простір в 3072 Мбайт

• Порти I / O (4x8 + 1x4)

• Телефонний CODEC інтерфейс з FIFO на 16 байт

• Програмований контроллер LCD (halfVGA - 640x240) з підтримкою DMA

• Повнодуплексний UART з двома 16-розрядними FIFO і логікою протоколу IrDA

• Синхронний послідовний інтерфейс

• Два 16-розрядних таймера / лічильника і сторожовий таймер

• Годинник реального часу з компаратором

• Два інтерфейсу з DC-DC перетворювачами

• Корпус PQFP з 208 виводами

Таким чином, на основі ядер ARM7 реалізуються досить потужні і складні прилади, що по своїй продуктивності наближаються до продуктивності робочих

станцій нещодавнього минулого, що забезпечується високопродуктивним RISC ядром і потужною 32-розрядною ARM системою команд.

Необхідно відзначити, що програми, підготовлені навіть для досить ефективної 32-розрядної ARM системи команд, вимагають пам'яті значного обсягу, що в свою чергу призводить до зростання загальної вартості системи. Фахівці фірми ARM запропонували рішення цієї проблеми, розробивши і впровадивши технологію Thumb, технологію, що дозволяє істотно скоротити обсяг кодів, необхідних для реалізації тієї ж програми, що виконується на 32-розрядній ARM системі команд. До теперішнього часу ця технологія вважається кращою з технологій, що використовують стислі системи команд.

Існує декілька підходів, що вирішують проблему розміру коду:

• Написання коду вручну на асемблері

Для отримання мінімального розміру коду програміст може писати коди вручну - на асемблері. Однак, цей процес може зайняти непрактично великий проміжок часу і в результаті може бути отримано код, який важко підтримувати і який є всього лише на 10-20% більш компактнішим ніж код, отриманий при використанні компілятора. Корінь проблеми, неефективність коду, все ще не вирішується.

• Використання поліпшеного компілятора

Технологія компілювання може поліпшити код, але знову таки меншим розмір коду буде при ручному кодуванні на асемблері.

• Використання компресованого коду

Одним з варіантів може бути використання деякої форми стисненого коду, який розгортається під час виконання. Однак, швидка декомпресія, яка не буде знижувати продуктивність процесора при виконанні цього коду, досить складна і вимагає використання додаткових ресурсів системи.

3. Концепція Thumb

Технологія Thumb - додаткове розширення до архітектури ARM. Система команд Thumb містить 36 команд, похідних від стандартної 32-розрядної

системи команд ARM, перекодованих в 16-розрядні коди. Такий підхід забезпечив дуже високу щільність коду, оскільки команди Thumb складають половину ширини формату команд ARM. У процесі виконання ці нові 16-розрядні Thumb коди декомпресуються процесором у відповідні еквівалентні команди ARM, які потім і виконуються ядром ARM звичайним способом.На рис.1.4 проілюстровано зв'язок ARMкоманд та команд Thumb, де команди Thumb розглядаються, як кодована підмножина системи команд ARM.

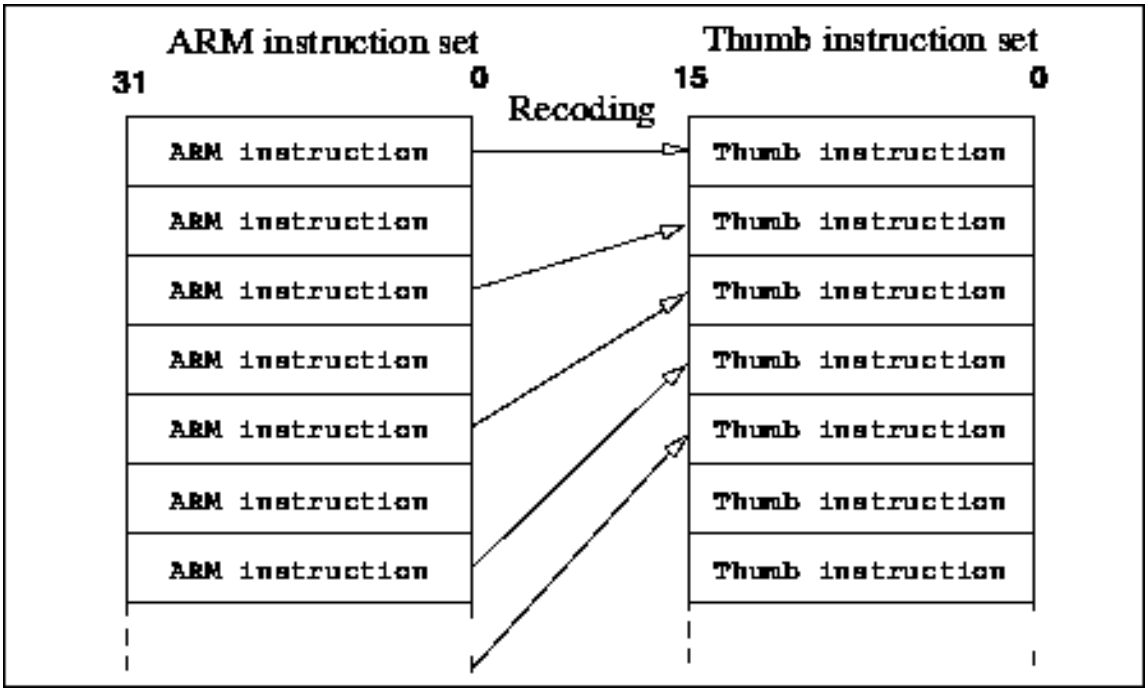


Рис.1.4. Команди Thumb як кодована підмножина системи команд ARM

Технологія Thumb - це не тільки змішана система команд. Thumb-орієнтовані ядра мають дві окремих системи команд - унікальнаперевага, що дозволяє розробнику використовувати всю потужність 32-розрядної системи команд ARM при використанні переваг малого розміру коду системи команд Thumb. Той факт, що дві системи команд є абсолютно окремими, говорить про те, що засобидекодуючої логіки також надзвичайно прості, що в свою чергу, зберігає малим розмір кристала і зберігає найкраще співвідношення в галузі продуктивність / споживання.

- Розмір і критичні до продуктивності підпрограми

Так як Thumb-орієнтовані ядра здатні виконувати і стандартну ARM систему команд і нові команди Thumb, розробник, при переході від підпрограми до підпрограми, може знаходити компроміс між розміром коду і продуктивністю, готуючи критичні до розміру підпрограми в коді Thumb і критичні до продуктивності підпрограми в кодах ARM.

- 32-розрядна RISC продуктивність

Thumb-орієнтовані ядра типу ARM7TDMI мають повну 32-розрядну архітектуру ARM, так що розробник зберігає 32-розрядну продуктивність RISC архітектури. Комбінація двох систем команд, що виконуються на 32-розрядному Thumb-орієнтованому ядрі, забезпечує ефективне вирішення проблеми великих розмірів коду та проблеми невисокої продуктивності 16-розрядних систем.

- Поліпшення щільності коду на 30%

Отримані до теперішнього часу результати показали поліпшення щільності коду на 30%, у порівнянні з кодом ARM, що дозволяє вважати Thumb-орієнтовані процесори кращими по щільності коду в порівнянні і з традиційними CISC процесорами.

- Підтримка півслів

Крім введення нових Thumb команд, фірма ARM додала до систем команд і ARM і Thumb підтримку формату півслів (16-розрядних даних). Отже архітектура ARM тепер повністю підтримує 8, 16 і 32-розрядні дані. Були додані і для Thumb і ARM ядер операції зі знаками для підтримки ними 8 і 16-розрядних операцій з даними із знаками.

- Засоби розробки

Комплект засобів розробки програмного забезпечення ARM також був розширений, щоб забезпечити підтримку розробки кодів Thumb. Комплект дозволяє програмісту писати і розміщувати в пам'яті системи коди ARM, коди Thumb або усі разом.

На рис. 1.5 зображена послідовність етапів проектування програмного забезпечення для Thumb-орієнтованого ядра.

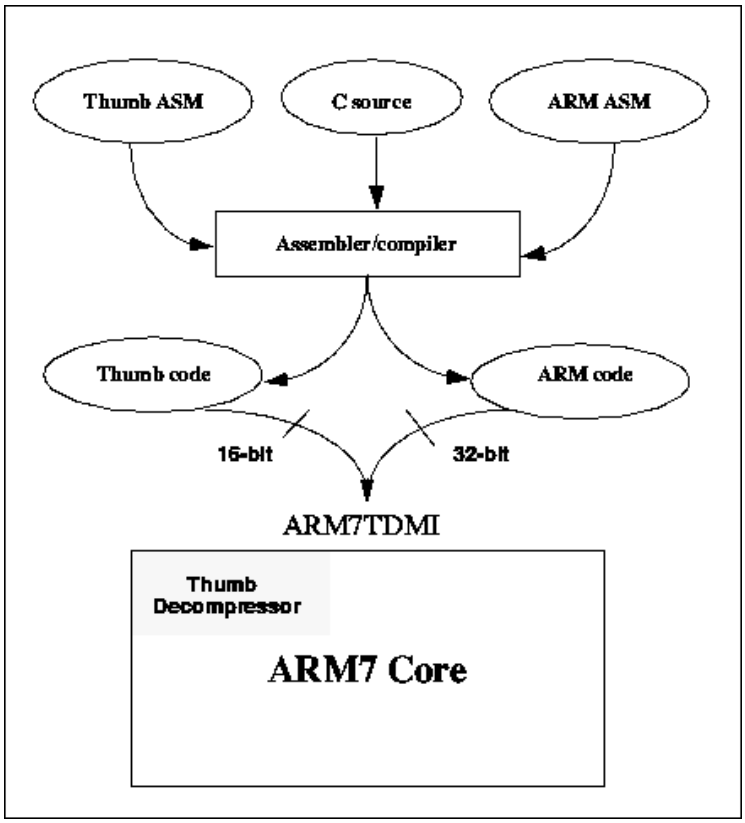


Рис. 1.5. Послідовність етапів проектування програмного забезпечення для Thumb-орієнтованого ядра

Так як Thumb-орієнтовані ядра - просто розширення архітектури ARM, розробник може компілювати коди Thumb, коди ARM або суміш обох. Ця сумісність вихідного тексту між Thumb-орієнтованими ядрами і ядрами ARM, забезпечує безпроблемний шлях до майбутніх оновлень до 32-розрядних систем, які вже знаходяться в експлуатації.

Крім того, простота реалізації технології Thumb гарантує, що перспективні ядра ARM із ще більш високою продуктивністю також будуть з Thumb-орієнтованими можливостями.

На рис. 1.6 відображені прикладні області застосування для Thumb-орієнтованого ядра.

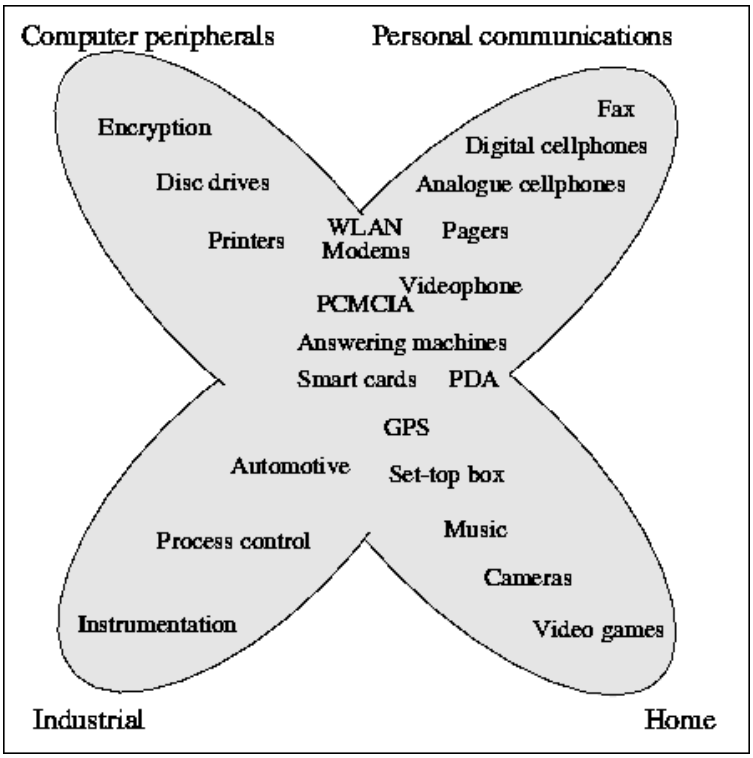


Рис.1.6. Прикладні області для Thumb-орієнтованого ядра

Нижче наведені три приклади конфігурації системи, що використовують ядро ARM7TMI.

Приклад 1. Варіант конфігурації мікроконролеру з ядром ARM7TDMIта 16-розрядною системою пам’яті (рис.1.7).

Переваги цієї системи в використанні зовнішньої шини і пам'яті вузького формату, що відповідає вимогам недорогих вбудовуваних застосувань.

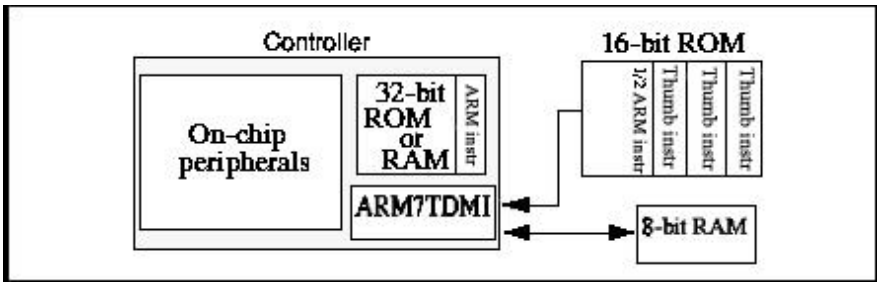


Рис. 1.7. Недорогий контролер з 16-розрядною системою пам'яті

У контролері інтегровані задані замовником вбудовані периферійні пристрої та невелика за обсягом швидка 32-розрядна ROM або RAM пам’ять, що використовуються для зберігання критичного до швидкодії коду. Коли Thumb-

орієнтоване ядро перемикається в стан ARM для отримання максимальної продуктивності, наприклад, обробки переривання, коди ARM виконуються з цієї області швидкодіючої пам'яті. Зовнішня 16-розрядна ROM використовується для зберігання кодів і констант, а 8-розрядна RAM містить надоперативнідані.

Приклад 2. Варіант конфігурації мікроконролеру з ядром ARM7TDMIта 32-розрядною ROM пам’яттю (рис.1.8).

Ця конфігурація показує, як Thumb-орієнтоване ядро може використовуватися з повільною, недорогою 32-розрядною ROM пам’яттю.

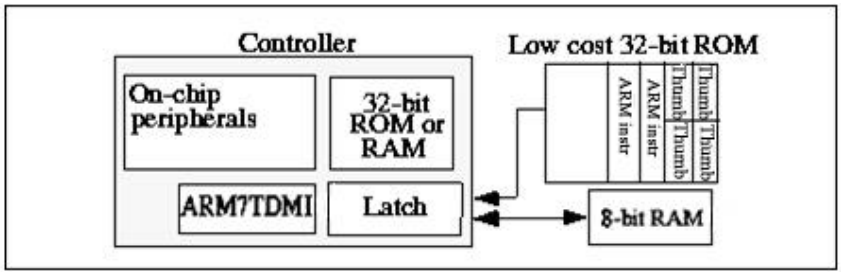


Рис.1.8. Система з недорогою 32-розрядної ROM пам’яттю

У даній конфігурації в ROM зберігається суміш підпрограм 32-розрядного коду ARM, з однією командою на 32-розрядне слово, і підпрограм коду Thumb з двома командами на кожне слово. Кожна зовнішня вибірка вибирає або 32-розрядну ARM команду або дві 16-розрядні команди Thumb. Команди ARM надходять в основний конвеєр звичайним способом. Однак, в стані Thumb, в той час коли одна команда Thumb надходить в конвеєр, інша зберігається в 16-розрядної комірці, яка є буфером вибірок команд з попередженням. При наступній вибірці, ця збережена команда негайно стає доступною ядру.

Моделювання показало, що в цій конфігурації з 200 нс ROM, технологія Thumb перевершує за швидкодією стандартне ядро ARM на10 - 20%, залежно від тактової частоти процесора та коду. Це пов'язано з тим, що ядру ARM для вибору кожної команди з ROM необхідний стан очікування, в той час як Thumb-орієнтоване ядро перебуває в стані очікування тільки один раз на кожні дві команди.

Приклад 3. Варіант конфігурації мікроконролеру з ядром ARM7TDMIта високопродуктивною 32-розрядною системою (рис.1.9).

Дана конфігурація представляє останнє Thumb-орієнтоване рішення перед переходом до стандартного ARM ядра, що забезпечує найвищу продуктивність 32-розрядної системи.

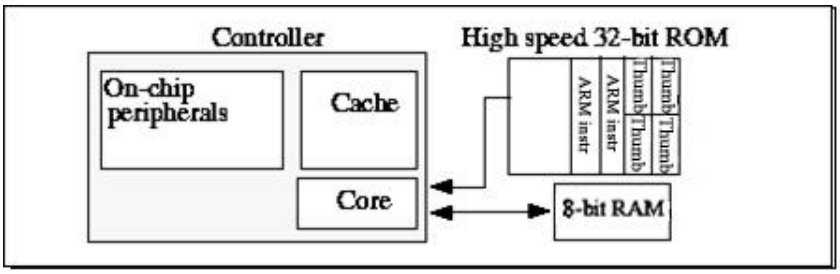


Рис. 1.9. Високопродуктивна 32-розрядна система

При використанні швидкодіючої ROM і вбудованому кеш, ця система забезпечує найвищу продуктивність для Thumb-орієнтованого ядра, оскільки 32-розрядні ARM команди можуть виконуватися безпосередньо з швидкодіючої пам'яті. Розмір коду і вартість системи звичайно більша ніж у недорогих 16-розрядних шини і системи пам'яті.

4. Особливості роботи першогоThumb-орієнтованого ядра ARM7TDMI

Першим Thumb-орієнтованим ядром стало ядро ARM7TDMI. Це ядро сімейства ARM7 має наступні особливості:

• ВбудованумакрокоміркуEmbeddedICE ™, що підтримує налагодження вбудованого ядра

• 32-розрядний апаратнийперемножувачем

• декомпресор Thumb

• 32-розрядна продуктивність в 8 - і 16-розрядних додатках.

Ядро ARM7TDMI поповнило стандартний ряд 32-розрядних ядер ARM. Ядро використовується як ліцензійна макрокомірка ASIC ARM, призначена для використання при створенні стандартних приладів спеціального призначення.

Нижче (Рис. 1.10) наведена блок-схема ядра ARM7TDMI, яке далі буде розглянуто більш детально.

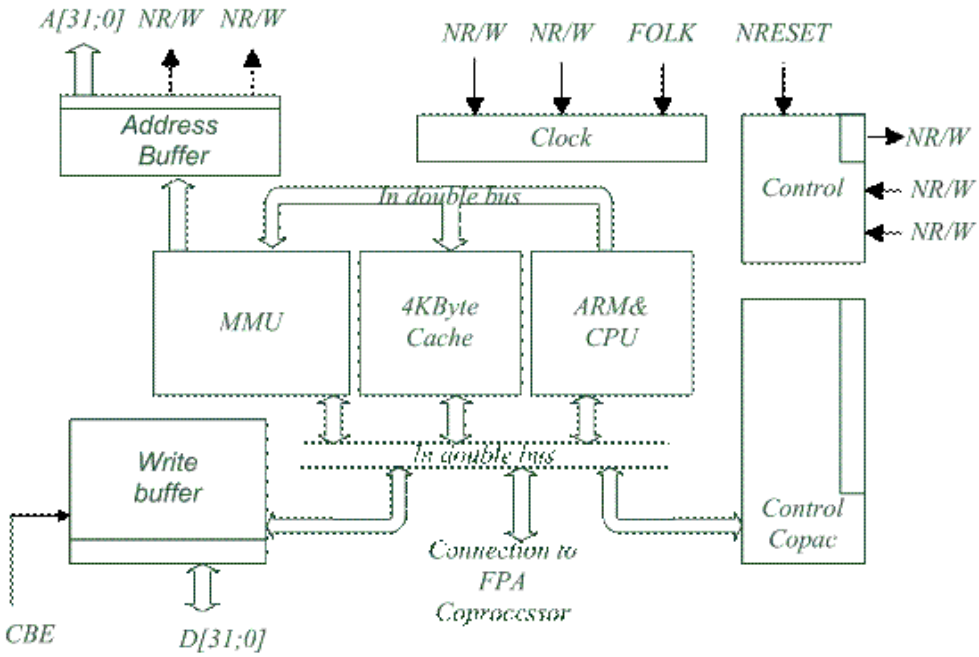


Рис. 1.10. Блок-схема ядра ARM7TDMI

Ядро ARM7 містить

· ARM7 CPU - 32- розрядний RISC процесор (32-розрядні шини даних і адреси);

· Cache – спільна кеш-пам‘ять команд і даних ємністю 4 кБайта;

· MMU – пристрій керування памяттю (для організації віртуальної пам'яті, для розбивання пам'яті на сторінки);

· AddressBuffer – буфер адреса ;

· WriteBuffer – буфер запису;

· Clock – тактовий генератор (25 МГц);

· Control – схема керування і синхронізації ;

· Control Coproc – схема керування під‘єднанням співпроцесорів.

На рис.1.4 наведено основні сигнали (виводи ВІС) ядра

· A[31:0] - шина адреси;

· D[31:0] - шина даних (використовується для передачі даних між процесором і зовнішньою пам'яттю);

· CBE - Дозвіл шини даних;

· nRW - Інверсний сигнал читання/запису. Якщо процесор виконує цикл читання, то даний сигнал має НИЗЬКИЙ рівень;

· ABE - Дозвіл шини адреси (Драйвери шини адреси відключаються переходом у високоімпедансний стан, якщо даний сигнал прийняв НИЗЬКИЙ рівень).

· MCLK - Вхід синхронізації пам'яті. Основна синхронізація для операцій процесора і доступу до пам'яті.

· nWAIT- Інверсний вхід вставки станів чекання Подачею НИЗЬКОГО рівня процесор продовжує доступ до периферійних пристроїв або пам'яті зниженої швидкодії протягом визначеної кількості циклів MCLK. Сигнал nWAIT усередині процесора піддається операції логічного "I" з MCLK і повинний змінятся тільки, коли MCLK має НИЗЬКИЙ рівень. Якщо nWAIT не використовується, то на нього необхідно подати ВИСОКИЙ рівень;

· SYNC - Синхронізовані переривання. Необхідно установити ВИСОКИЙ стан, якщо nIRQ і nFIQ синхронізовані з процесорною синхронізацією або НИЗЬКИЙ стан для асинхронних переривань;

· nRESET - Інверсний вхід скидання;

· nMREQ- Інвертований запит пам'яті;

· nFIQ - Інвертований запит швидкого переривання. Використовується для переривання (після активізації цієї функції) процесора подачею на цей вхід НИЗЬКОГО рівня. Сигнал є чутливим до рівня і повинен утримуватися в НИЗЬКОМУ стані до одержання підходящого відгуку від процесора. nFIQ може бути синхронізованій або асинхронним стосовно MCLK, у залежності від стану ISYNC;

· nIRQ - Інвертований запит переривання Діє аналогічно nFIQ, але з більш низьким пріоритетом. Робота процесора переривається подачею на цей вхід низького рівня, якщо відповідний дозвіл активно.

Всі блоки процесора з‘єднані по внутрішній шині адреси (Internal Address Bus) та даних (Internal Data Bus), до яких може бути під‘єднані співпроцесори (Connection to FPA Coprocessor). Співпроцесор підключається до тієї ж шини даних, до якої підключається процесор ARM7TDMI у системі, і відслідковує конвеєр у процесорі ARM7TDMI. Це означає, що співпроцесор може дешифрувати інструкції в потоці інструкцій і виконувати ті з них, що він підтримує. Кожна інструкція просувається по конвеєру одночасно, як у ядрі ARM7TDMI, так і в співпроцесорі.

Thumb-версії стандартних ядер фірми ARM, вже розроблених і тих, що знаходяться в розробці, здатні принести ще більш високу продуктивність у 8 і 16-розрядний світ вбудованого управління. Технологія ущільнення коду Thumb в поєднанні з унікальною продуктивністю StrongARMзабезпечить унікальні по продуктивності рішення для застосувань з вбудованим управлінням, що вимагають продуктивності на рівні робочої станції.

Основний додаток до архітектури ARM, що забезпечує підтримку системи команд Thumb - декомпресор Thumb. Першим ядром ARM, яке містило декомпресор, стало ядро ARM7TDMI. На рис.1.11 зображена конвеєрна вибірка, декодування і виконання

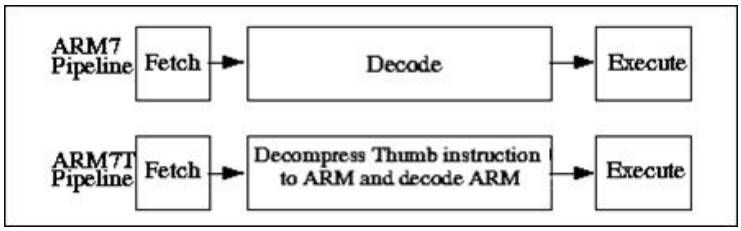


Рис. 1.11. Конвеєрна вибірка, декодування і виконання

І ARM7 і ARM7TDMI ядра в одному тактовому циклі використовують 3-рівневий конвеєр з фазами вибірки, декодування і виконання. Потік команд через кожен рівень конвеєра управляється високими і низькими фазами тактового

сигналу. У ядрі ARM7TDMI невживана фаза тактового сигналу використовується для декомпресування команд Thumb в каскаді декодування. Отже, в одному тактовому циклі проводиться і декодування і виконання команди, не потрібно ніяких додаткових непродуктивних витрат синхронізації. На рис. 1.12 зображено процес Thumb-декодування і декомпресії.

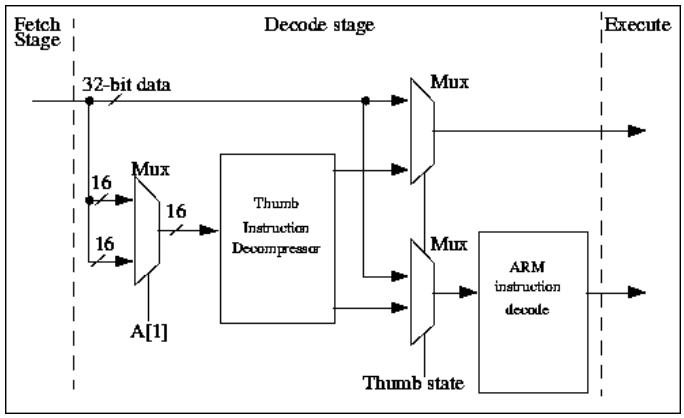


Рис. 1.12. Thumb-декодування і декомпресія

Команди ARM, що надходять з каскаду вибірки конвеєра, проходять через декодер ARM і активують сигнали управління старшими і молодшими бітами операційного коду. Старші біти операційного коду описують тип команди, яка виконується, тоді як молодші біти визначають деталі команди типу визначення регістрів або операнда.

У Thumb-стані мультиплексори направляють Thumb-команди через логіку декомпресії Thumb, розгортаючи команду Thumb в її еквівалент ARM команди. Потім команда ARM виконується в нормальному режимі. Це легко зрозуміти на прикладі (Рис.1.13):

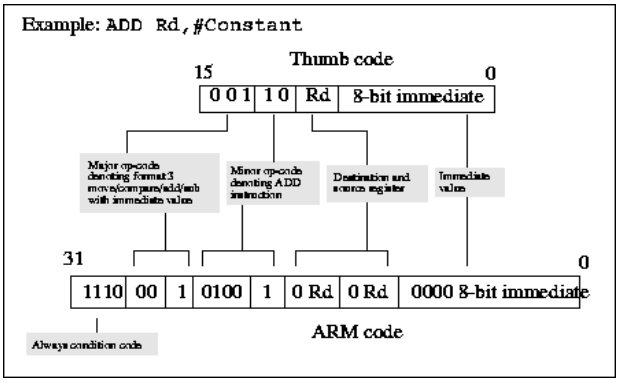


Рис.1. 13. Трансляція команди ADD Thumb в команду ADD ARM

Як видно з рис.1.13, старший опкод команди Thumb поміщається в команду ARM, а молодший транслюється через довідкову таблицю.

У команді ARM завжди присутній код умови, що визначається з старшим операційним кодом.

Старший опкод визначає маршрут операнда від операційного коду Thumb до операційного коду ARM. Визначники регістрів розширюються додатковим нулем від опкодаThumb (3 біта) до 4 бітів, оскільки ця команда Thumb звертається тільки до регістрів ARM R0-R7. Значення константи також розширюється нулями, визначаючи 8-розрядну константу в опкоді ARM.Це рішення застосовне на будь-якому ядрі ARM і буде використовуватися в перспективних ARM архітектурах.

При створенні 16-розрядних операційних кодів, у систему команд Thumb були введені деякі обмеження. Найбільш очевидні: зменшене число регістрів, доступних при виконанні коду Thumb. Замість п'ятнадцяти 32-розрядних регістрів загального призначення (GPR) плюс PC процесора ARM, програміст має доступ до восьми GPR регістрам, покажчику стека, регістру зв'язків (LinkRegister) і PC. В системі команд Thumb, вісім GPR регістрів (R0-R7) називаються Lo набором регістрів. Інші регістри ARM (R8-R15) позначаються як Hi регістри. Програміст в Thumb режимі обмежений доступом до Hi регістрам через команди переміщення, порівняння і команду ADD, що надають йому деяку локальну швидку тимчасову середу зберігання. Перемикання між ARM і Thumb кодами не впливають на вміст GPR регістрів. На рис. 1.14 показано відображення Thumb регістрів стану на регістри стану ARM.

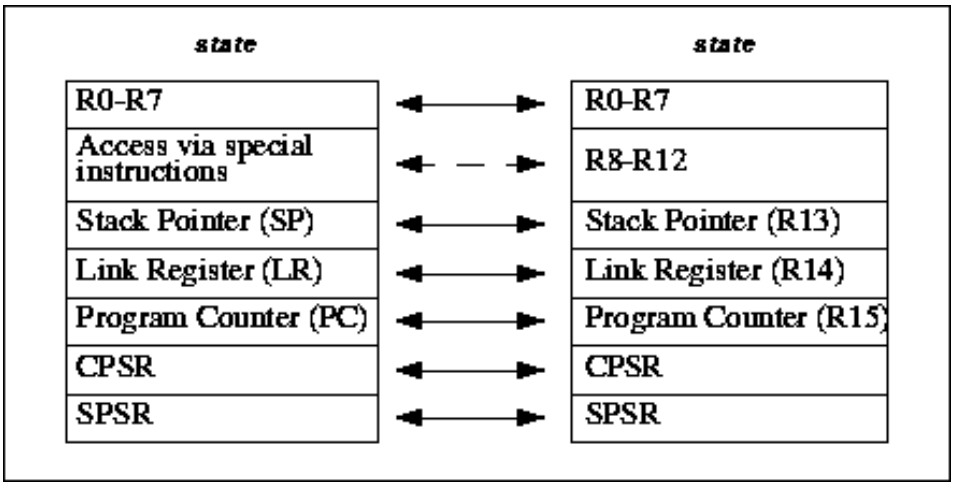


Рис. 1.14. Відображення Thumb регістрів стану на регістри стану ARM

Ядро ARM7TDMI реалізує 7 видів переривань, що перераховані нижче в порядку зниження їхнього пріоритету:

1. Reset - реалізується при подачі сигналу запуску на зовнішній вивід процесора. Процесор переходить у режим Supervisor і починає виконання програми з адреси, розміщеного в першій комірці пам'яті (адреса комірки 0x00000000).

2. Dataabort - помилка при звертанні до даних (фіксується контролером переривань). Процесор переходить у режим Abort.

3. FIQ - реалізується при подачі сигналу переривання на один з виводів nFIQ. Процесор переходить у режим FIQ. При надходженні даного переривання в регістровому банку відбувається переключення 7 регістрів R8-R14, що дозволяє скоротити або виключити операції збереження вмісту регістрів у стеці. За рахунок цього перехід до обробки переривання відбувається швидше.

4. IRQ - виникає при подачі сигналу переривання на один з виводів nIRQ. Процесор переходить у режим IRQ.

5. Prefetchabort - помилка при вибірці команди (фіксується контролером переривань). Процесор переходить у режим Abort.

6. Undefinedinstruction - вибірка неправильного коду команди. Процесор переходить у режим Undefined.

7. Softwareinterrupt - програмне переривання по команді SWI. Процесор переходить у режим Supervisor. Програмні переривання використовуються, як правило, для виклику функцій ОС.

Ядро ARM7 TDMI може функціонувати в двох станах: ARM і THUMB. У стані ARM процесор виконує 32-розрядні команди, у стані THUMB — 16-розрядні команди. Перехід процесора зі стану ARM у Thumb і навпаки реалізується за допомогою команди BX.

У стані ARM процесорможефункціонувати в одному з наступнихрежимів:

1. User – режим виконання програм користувача (прикладних).
2. Supervisor - робота під керуванням операційної системи (ОС), що оперує даними, недоступними програмам користувача.
3. System - режим виконання системних програм, при якому ОС працює з даними користувача.
4. IRQ - режим обробки переривань загального призначення, у котрий попадає процесор при надходженні запиту переривання нижчого рівня IRQ.
5. FIQ (Fast IRQ) - режим швидкої реакції на переривання, у котрий попадає процесор при надходженні запиту вищого рівня FIQ.
6. Abort - аварійний режим, що вводиться після аварійної ситуації (помилці звертання до пам'яті) наприклад, звертання по неіснуючій адресі, спробі запису в ПЗП, тощо. Аварійна ситуація фіксуються контролером переривань, який видає процесорному ядру запит Abort.
7. Undefined - невизначений режим. Цей режим реалізується при вибірці неправильного коду команди.

Усі режими, крім режиму користувача, спільно називаються привілейованими режимами. Привілейовані режими використовуються для обслуговування переривань і виняткових ситуацій, а також для доступу до захищених ресурсів

Регістрові моделі для кожного режиму стану функціонування ARM наведено на рис.1.15.

Рис. 1.15. Регістрова модель у стані ARM

Банкіровані регістри (рис.1.16) - це такі, які доступні у залежності від поточного робочого режиму процесора. Уміст банкірованого регістра запам'ятовується при змінах робочих режимів.

У режимі FIQ є сім банкірованих регістрів у позиціях r8-r14 (r8_fiq-r14_fiq).

У набір регістрів у стані ARM входять 16 32-розрядних регістрів r0...r15. і регістр стану CPSR (рис.1.14). Регістри r0...r13 є регістрами загального призначення і можуть використовуватися як для збереження даних, так і для збереження адреси. Регістри r14 і r15 виконують наступні спеціальні функції: Регістр 14 використовується як регістр зв'язку (LR - LinkRegister) підпрограми, він приймає копію регістру r15 при виконанні інструкції переходу за посиланням (BL). В всіх інших випадках регістр r14 може використовуватися як регістр загального призначення. Відповідні банкіровані регістри r14_svc, r14_irq, r14_fiq, r14_abt і r14_und аналогічним образом використовуються для запам'ятовування значень повернення при виникненні переривань і виняткових

ситуацій або при виконанні інструкції BL усередині процедур обробки переривань або виняткових ситуацій. Регістр 15 зберігає значення PC. Для організації стека рекомендується використовувати регістр R13 (SP - StackPointer) як покажчик стека

Регістр стану CPSR містить наступні біти (рис. 1.16):

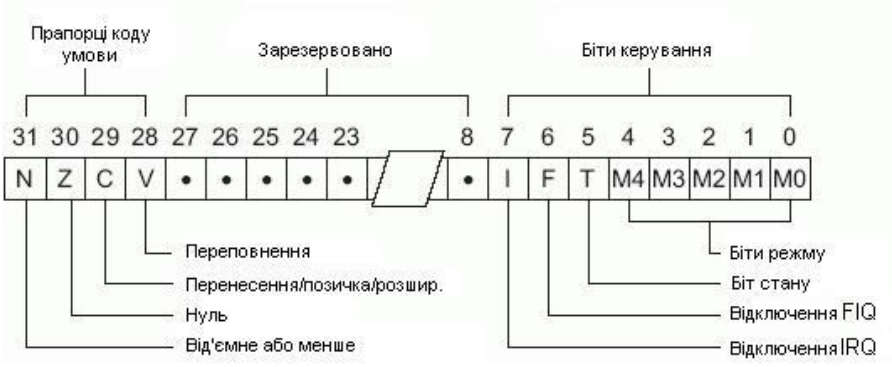


Рис. 1.16. Регістр стану CPSR

· M4-0 - визначають режим роботи процесора відповідно до табл. 1;

· T - задає стан процесора: ARM (при T = 0) або THUMB (при T = 1);

·I, F - маскують (забороняють обробку) переривань IRQ і FIQ відповідно

·N, Z, C, V - є ознаками знака (N), нуля (Z), перенесення (C) і переповнення (V), значення яких установлюються відповідно до результату чергової операції. У таблиці M4-M0 зображено відповідність конфігурації бітів M4-0 режимам процесора. Запис нового вмісту в регістр CPSR можливий у всіх режимах, крім режиму User.

Таблиця 1.1. Режими процесора

M4-0					Режим
1	0	0	0	0	**User**
1	0	0	0	1	**FIQ**
1	0	0	1	0	**IRQ**
1	0	0	1	1	**Supervisor**
1	0	1	1	1	**Abort**
1	1	0	1	1	**Undefined**
1	1	1	1	1	**System**

Стан Thumb відрізняється від ARM розрядністю команди - вона містить 16 біт. Убудований декодер переводить команди, записані по системіThumb, у команди ARM. Через компактністьвикористовуванихформатів, система команд Thumbмає ряд особливостей:

Набір регістрів у стані Thumb скорочений - він є підмножиною регістрового банку в стані ARM. Програміст має доступ до:

- 8 регістрів загального призначення r0-r7
- Лічильника програм PC
- Покажчика стека SP
- Регістру зв'язку LR
- Регістру поточного стану програми CPSR.

У кожнім привілейованому режимі є банкіровані регістри SP, LR і SPSR. Набір регістрів у стані Thumb показаний на рис. 1.17.

Рис. 1.17. Регістрова модель у стані ***Thumb***

У форматі більшості команд Thumb під номер регістра відведено тільки 3 біти, тому пряме звертання можливе тільки до регістрів R0-R7. Регістри R8-R12 доступні тільки через спеціальні команди завантаження. Регістри SP і LR виконують певні функції покажчику стека і регістр зв'язку. У стані Thumb є спеціальні команди, орієнтовані на роботу з цими регістрами - завантаження в стек відбувається тільки через SP, причому адреса повернення з підпрограми зберігається тільки в LR. Скорочення довжини команд уплинуло також на розрядність операндів. Безпосередні дані скорочені до 8 біт, адресні зміщення в більшості команд - до 5 біт. Однак оброблювані дані зберігають 32-розрядний формат.

При розробці інтерфейсу пам'яті розробники орієнтувалися на забезпечення реалізації потенційно можливої продуктивності без підвищення вартості самої пам'яті. Інтерфейс пам'яті ідеально підходить для організації взаємодії як з вбудованою на кристал пам'яттю, так і з зовнішньою пам'яттю, з

блоками Flash пам'яті, що дозволяє реалізувати внутрішньосистемне програмування, захист пам'яті, знизити час виходу на ринок, скоротити загальну вартість системи.

Інтерфейс з пам'яттю у процесора ARM7TDMI організовується наступними основними елементами:

• 32-розрядною шиною адреси, що визначає адресу комірки пам'яті, яку необхідно використовувати.

• 32-розрядноюдвонаправленою шиною передачі даних D [31:0], плюс двома окремими односпрямованими шинами даних DIN [31:0] і DOUT [31:0], через які переміщаються команди і дані. Дані можуть мати формат слова, півслова або байта.

• Сигналами управління, визначальними, наприклад, формат переміщуваних даних і напрямок їх передачі і, крім того, рівень пріоритету.

Апаратні розширення налагодження ARM7TDMI забезпечують розгорнуті можливості налагодження, що полегшують розробку користувальницького прикладного програмного забезпечення, операційних систем, і самих апаратних засобів. Апаратні розширення налагодження дозволяють зупиняти ядро або при вибірці заданої команди (в контрольній точці) або при зверненні до даних (в точці перегляду), або асинхронно - за запитом налагодження.

У цих точках, через JTAG послідовний інтерфейс, може бути досліджено внутрішній стан ядра ARM7TDMI, що знаходиться в стані налагодження, і зовнішній стан системи. По завершенні дослідження стану ядра і системи можуть бути відновлені і продовжено виконання програми.

Йдучи назустріч запитам фірм-розробників фірма ARM розширила номенклатуру функціональних макроядер на основі ядра ARM7TDMI (будемо називати макроядром деяке ядро, в даному випадку ARM7TDMI, зі схемотехнічно інтегрованими до нього додатковими, які розширюють його можливості) і тепер в сімейство ARM7 Thumb входять: ядро ARM7TDMI, макроядра ARM710T, ARM720T і ARM740T, і ядро ARM7TDMI-S.

Основні характеристики чотирьох перших процесорів наведені в таблиці 1.2.

Таблиця 1.2. Характеристики процесорів ARM7.

Ядро CPU	Площакристалу	Споживання (mW/MHz)	Тактова частота	Продуктивність	Ядро CPU	Кеш	Керування пам’яттю
ARM7TDMI ARM RISC ядрос Thumb иEmbeddedICE	1,0 мм^{2}при 0,25мкм 2,1 мм^2 при 0,35 мкм 4,8 мм^2 при 0,6 мкм	Піковое: 1,2 Середнє: 0,6 Idle: < 100 мкВт, при 3,3 В, CMOS 0,35 мкм	66 МГц при нормах 0,35 µм, CMOS	0,9 MIPS/MHz 59 MIPS при 66 МГц	N/A	N/A	N/A
ARM710T Кешированепроцесорнемакроядро	5,8 мм^{2}при 0,25 мкм 11,7 мм^{2}при 0,35 мкм	Пікове: 3,6 Середнє: 1, 8 Idle: < 100 мкВт при увімкненому кеш, 3,3 В, 0,35мкм CMOS	59 МГц при нормах 0,35мкм CMOS	53 MIPS при 59МГц	ARM7TDMI	8 Кбайт єдиний кеш	MMU з повною підтримкоювірту альної пам’яті
ARM740T Кешированепроцесорнемакроядро	4,9 мм^{2}при 0,25 мкм 9,8 мм^2 при 0,35 мкм	Пікове: 3,5 Середнє: 1,6 Idle: <100 мкВт при увімкненому кеш, 3,3 В, 0,35мкм CMOS	59 МГц при нормах 0,35мкм CMOS	53 MIPS при 59МГц	ARM7TDMI	8 Кбайт єдиний кеш	Проста конфигурація памятітазахисту
ARM720T Кешированепроцесорнемакроядроз MMU для WindowsCE	5,8 мм^{2}при 0,25мкм 11,7 мм^{2}при 0,35 мкм	Пікове: 3,6 Середнє: 1,8 Idle: < 100 мкВт при увімкненому кеш, 3,3 В, 0,35мкм CMOS	59 МГц при нормах 0,35мкм	53 MIPS при 59МГц	ARM7TDMI	8 Кбайт єдиний кеш	MMU з повною підтримкоювірту альної пам’ятіташвидкого контекстного переключення

5. Вбудована системна шина AMBA

Під приладами класу "система-на-кристалі", в загальному випадку, маються на увазі прилади на єдиному кристалі до яких інтегровані процесор

(процесори, у тому числі спеціалізовані), деякий обсяг пам'яті, ряд периферійних пристроїв і інтерфейсів - тобто максимум того, що необхідно для вирішення завдань, поставлених перед системою. Фірмою ARM, окрім вже згадуваних ядер сімейств ARM7 і ARM7TDMI, розроблений набір макрокомірок периферійних компонентів, які фірма на основі ліцензійних угод надає замовникам. Периферійні компоненти фірми ARM, бібліотека яких отримала найменування PrimeCell, являють собою готові до застосування відпрацьовані програмні макрокомірки, при розробці яких зверталася увага на можливість багаторазового їх використання.Застосовуючи PrimeCell периферію розробник істотно заощаджує час і вартість розробки за рахунок концентрації зусиль на створенні саме системи на кристалі, а не на розробці спочатку необхідної периферії і лише потім системи. В даний час в бібліотеку входять: UART, контролери SDRAM, синхронні послідовні інтерфейси, годинник реального часу, аудіо кодеки, засоби I / O загального призначення, інтерфейси смарт карт, контролери кольорових LCD. Ведуться роботи по подальшому розширенню бібліотеки.

Продуктивність приладів класу "система-на-кристалі" в значній мірі залежить від ефективності взаємодії всіх встановлених компонентів і від ефективності їх взаємодії із зовнішнім, щодо приладу, світом. У першу чергу це пов'язано з відмінностями у швидкодії вбудованих компонентів, в особливостях організації інтерфейсів.

При розгляді макроядер була згадана шина AMBA (AdvancedMicrocontrollerBusArchitecture) - шина розроблена фірмою ARM для організації ефективної взаємодії компонентів приладів, побудованих на базі ядер фірми. Шина AMBA - стандартна вбудована ASIC шина забезпечує швидке модульне проектування систем при спрощенні багаторазового використання схемотехніки та тестів. ARM також забезпечує можливість використання бібліотеки PrimeCell периферії, яка відповідає AMBA стандарту і забезпечує просту розробки ASIC і ASSP.

Блок-схема шини AMBA в приладі типу персонального інформаційного пристрою (PDA), реалізованого на основі ядра ARM, макрокомірок бібліотеки PrimeCell і шини AMBA, представлений на рис. 1.18.

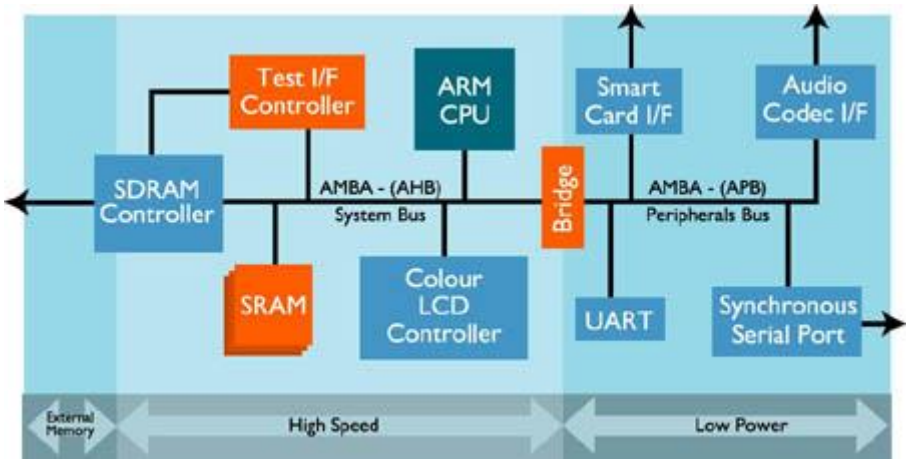


Рис. 1.18. Приклад приладу класу "система-на-кристалі", що використовує шину AMBA

Розвинута високопродуктивна шина (AdvancedHigh-performanceBus - AHB)

Шина AHB використовується у високопродуктивних системах класу "система-на-кристалі" і відповідає сучасним вимогам пропонованим процесом синтезу приладів з рівнем інтеграції система-на-кристалі.

• Синхронізація наростаючим фронтом кожного тактового сигналу

Використання кожного тактового імпульсу забезпечує максимальну продуктивність.

Тимчасова діаграма відповідає вимогам послідовності процесу синтезу.

• Працює в режимі з великою кількістю ведучих

Продуктивність системи оптимізується за рахунок поділу ресурсів між різними ведучими на шині, такими як головний процесор, контролери DMA, інші процесори або співпроцесори.

• Конвеєрні і пакетні пересилки

Конвеєрна робота забезпечує звернення до швидкодіючої пам'яті або периферії без використання режимів очікування, без неодружених циклів шини.

Пакетна робота дозволяє оптимально використовувати інтерфейс з пам'яттю за рахунок надання додаткової інформації про характер даних, що пересилаються.

• Підтримка поділу транзакцій

Забезпечує максимальне використання смуги пропускання системної шини за рахунок вивільнення її від повільних пристроїв на час завершення їх внутрішніх операцій.

• Можливість конфігурування в широкому форматі (формати від 32/64/128 до 1024 біт)

Підтримує застосування з широкоформатною вбудованою пам'яттю з інтенсивним обміном даними та широкою смугою частот.

Розвинута системна шина (AdvancedSystemBus - ASB)

Це оригінальна системна шина AMBA, розроблена на основі інтерфейсу ARM:

• Працює в режимі з безліччю провідних

• Забезпечує конвеєрні та пакетні пересилки

Прикладом використання шини ASB може служити блок-схема, представлена на рис. 1.19.

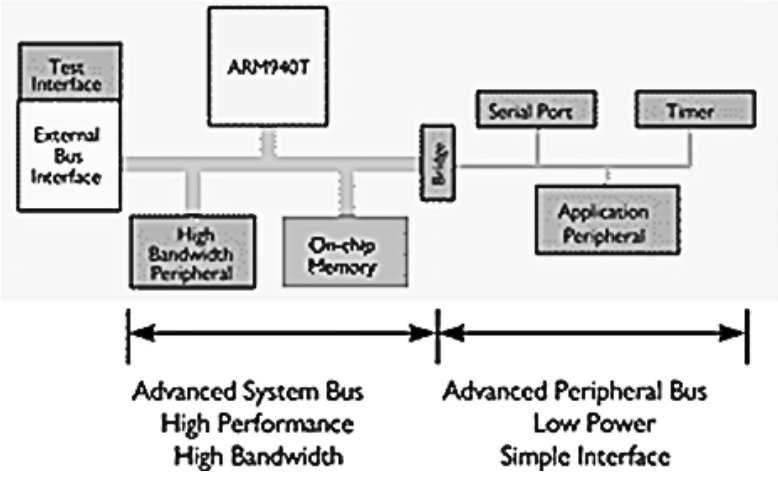


Рис. 1.19. Блок-схема приладу, реалізованого на основі ядра ARM940T.

Як і в раніше наведеному прикладі шина периферії з'єднується з системною шиною (в даному випадку ASB) за допомогою моста.

AMBA шина периферії

Розвинута шина периферії (AdvancedPeripheralBus - APB)

AMBA шина периферії призначена для організації інтерфейсу з вбудованими периферійними пристроями загального призначення, такими як таймери, контролери переривань, UART, порти I / O і т.п., і додатковими периферійними пристроями. З основною системною шиною шина периферії з'єднується мостом, який забезпечує розвантаження системної шини і знижує загальне споживання системи.

У відповідності з новою специфікацією AMBA Rev 2.0 Specification шина відповідає сучасним вимогам послідовності синтезу приладів класу "система-на-кристалі".

• Проста шина

Бесконвейерная архітектура

Просте використання - всі периферійні пристрої обслуговуються як ведені

Мала кількість використовуваних вентилів

• Мале споживання

Зниження завантаження основної системної шини за рахунок її ізоляції від периферії мостом

Сигнали на шині периферії активні тільки під час повільних пересилань периферії

6. Розвиток та перспективи

В основу процесорної лінійки Cortex компанія ARM поклала як нову архітектуру ARMv7, так і нову ідеологію маркетингу своєї інтелектуальної власності.

Архітектура ARMv7, будучи повністю сумісною з напрацюваннями попередниці ARMv6, включила до свого складу ряд якісно нових рішень,

покликаних істотно підвищити продуктивність процесорів ARM у відповідності зі все зростаючими вимогами нових інформаційних технологій.

До цих рішень в першу чергу відноситься технологія Thumb-2 - наступне покоління успішної системи ущільнення коду Thumb. Thumb-2 розширила набір 16-розрядних Thumb-команд і доповнила його поруч повноцінних 32-розрядних інструкцій. Саме тому Thumb-2, при щільності коду аналогічної Thumb, забезпечує істотний приріст продуктивності.

Розробивши єдину архітектуру ARMv7 для всієї лінійки Cortex, компанія ARM вирішила розділити варіанти процесорних ядерCortex по областях їх застосування. В результаті цього маркетингового кроку процесори лінійки Cortex були розділені на три класи.

1. Cortex-A (від application) - сімейство процесорів, орієнтованих на ринок споживчої електроніки і здатних вирішувати широкий спектр завдань, яким сучасні користувачі так люблять навантажувати свої гаджети. До складу лінійки Cortex-A увійшли процесори, назви більшості з яких у нинішньому році були у всіх на слуху, а назви деяких тільки готуються до цієї важливої місії. В даний час компанією ARM розроблений референс-дизайн процесорів Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9 і Cortex-A15.

2. Cortex-R (від realtime) - серія мікропроцесорів, оптимізованих для виконання обчислень в режимі реального часу. Скрізь, де потрібно відгук системи в строго певний час, процесори серії Cortex-R забезпечать його. Найбільш бажаними галузями застосування цієї процесорної лінійки є, звичайно ж, вбудовані рішення, такі як контролери промислового та медичного обладнання, автомобільна електроніка та комунікаційні системи.

Втім, і в споживчій електроніці Cortex-R знаходиться гідне застосування. Контролери жорстких дисків, процесори обробки сигналів у фото / відеомодулем смартфонів і цифрових камер, набирає силу «розумне» телебачення - ось тільки мала частина областей, де «реальний» Cortex може застосувати свої здібності.

3. Cortex-M (від eMbedded) - лінійка Cortex-процесорів, які прийшли на зміну 8 - і 16-розрядних мікроконтролерів вбудованих систем. Володіючи всіма достоїнствами і міццю архітектури ARMv7, процесори серії M є самими легковагими серед своїх Cortex-побратимів. Низьке енергоспоживання, мінімальне тепловиділення і маленькі габарити поряд з незвичайною продуктивністю дозволяють Cortex-M стати на чолі «розумною» побутової техніки, а також інтелектуальних контролерів в областях автомобільної електроніки та ігрових консолей.

Кожна наступна лінійка процесорів ARM підтримує технологічні рішення попередників і включає в себе нові технології.Еволюція процесорів та технологій ARM представлена на рис.1.20.

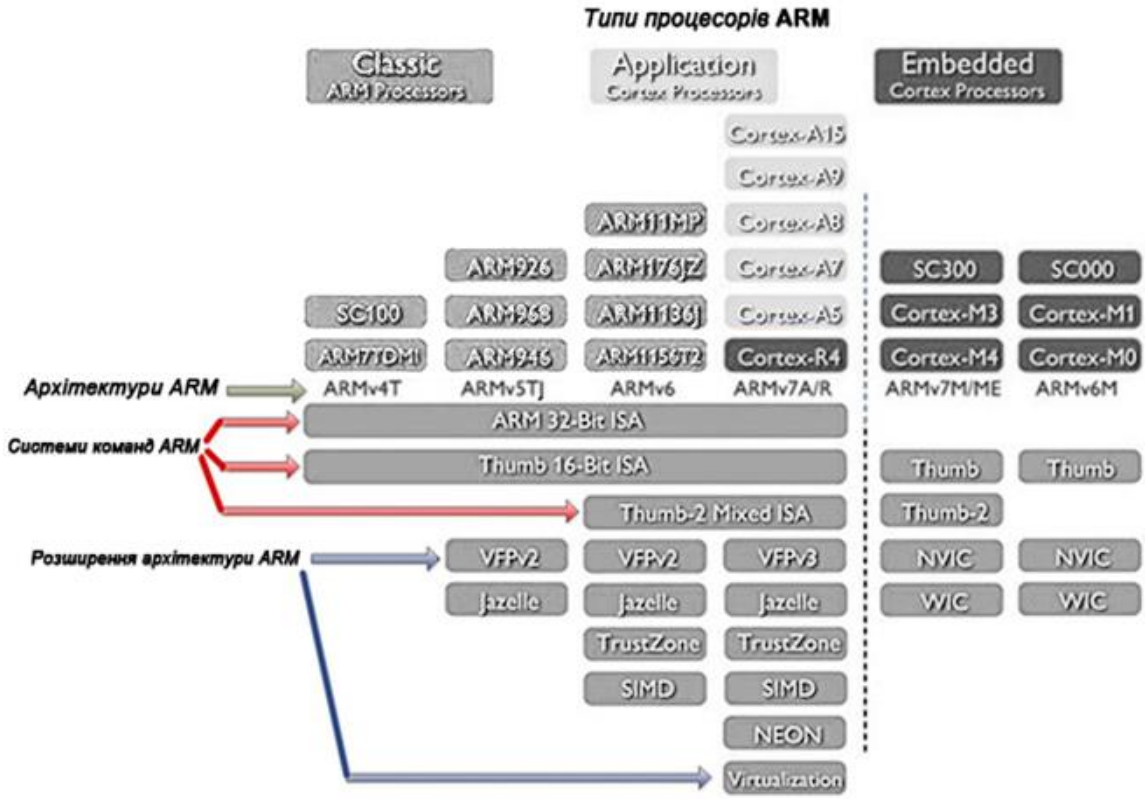


Рис. 1.20. Еволюція процесорів та технологій ARM.

Маркетологи ARM розділили всі процесори компанії на лінійки Classic і Cortex. Процесори Cortex, у свою чергу, діляться на рішення для прикладних (application) і вбудовуваних (embedded) рішень. Структура поділу процесорів ARM на класи представлена на рис. 1.21.

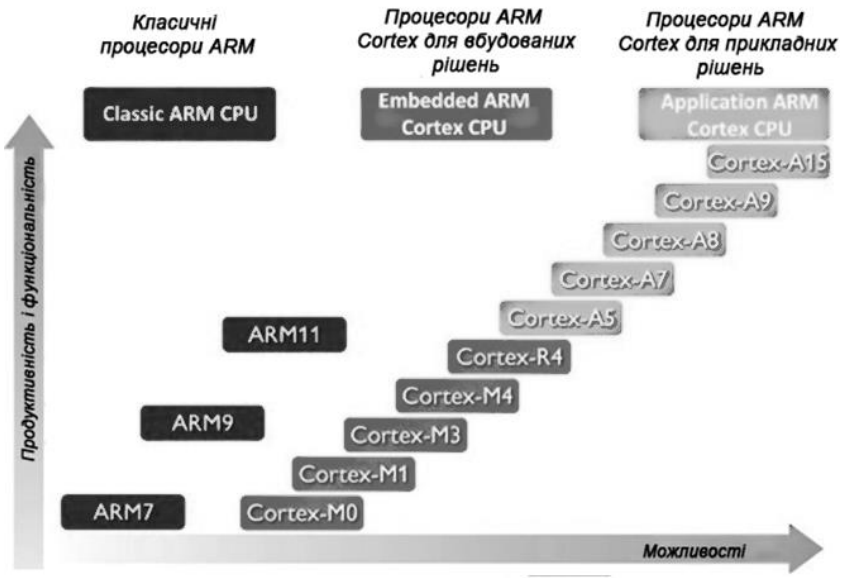


Рис. 1.21. Поділ процесорів ARM на класи.

Тема 3.2 Сімейство 32-бітних мікроконтролерів STM

Мікроконтролери серії STM32 побудовані на основані на ARMCortex-M4 за 90нм-технології і використовують запатентований STMicroelectronics'ом ARTAccelerator для досягнення найкращих результатів тестів серед мікроконтролерів на ядрі Cortex-M мікроконтролерів, досягаючи показників в 225 DMIPS / 606 CoreMark і працюючи з флеш -памятю на частоті 180 МГц. Інструкції DSP і модуль операцій з плаваючою крапкою дають можливість застосовувати дані контролери в широкому спектрі проектів. Динамічне споживання живлення дозволяє знизити споживання струму при виконанні коду з флеш-пам'яті до 140 мкА / МГц для STM32F401 (максимальна частота до 84 МГц) і до 238 мкА / МГц для STM32F42x / 43x, що працюють на частоті до 180 МГц. Мікроконтролери серії STM32 є результатом ідеального поєднання можливості управління МК в реальному часі і продуктивністю обробки сигналів, властивої сигнальним процесорам, доповнюючи таким чином лінійку контролерів STM32 новим класом пристроїв, сигнальними микроконтролерами (англ. Digital signal controller, DSC). Серія складається з п'яти класів пристроїв, які повністю сумісні по виводам периферії та програмному коду.

1. Архітектура мікроконтролера STM32F407VG, його характеристики.

Нова серія мікроконтролерів STM32 F4 є розширенням платформи STM32 заснованим на останній версії ядра ARM Cortex-M4. У цій серії з'явилися нові можливості у сфері обробки сигналів і більш швидкі за часом виконання операції (Рис.2.1).

Рис.2.1 Зовнішній вигляд STM32F407VG

Нові контролери STM32F4xx, крім високопродуктивного ядра ARM Cortex-M4, що працює на частоті до 180 МГц, володіють інтегрованим TFT-LCD-контролером, графічним прискорювачем Chrom-ART, підтримкою динамічної пам'яті SDRAM і послідовним аудіо-інтерфейсом SAI.

У новій серії STM32 F4 компанія додалися DSP інструкції, що виконуються за один такт, які відкривають для нового продукту двері на ринок цифрових сигнальних контролерів, покращуючи обчислювальну здатність і DSP-інструкції для особливо вимогливих в цьому плані пристроїв, таких як медичне обладнання, управління двигунами і охоронне устаткування. Забезпечуючи апаратну (повний збіг висновків) і програмну сумісність з серією STM32 F2, і в той же час, маючи більший обсяг SRAM, підвищену продуктивність і кращу завадостійкість периферії, серія F4 дозволить розробникам поліпшити кінцевий продукт, якщо їм необхідно більше обсягу

пам'яті, продуктивності або особливостей периферії. Також, якщо розробник використовує в своєму продукті два чіпи - MCU і DSP, то тепер він може об'єднати можливості цих двох чіпів в одному високопродуктивному сигнальному контролері.

STM32F407 має наступні перваги перед мікроконтролерами цієї серії: розширена периферія, на додаток до периферії STM32F405 доданий другий USB-OTG інтерфейс, інтегрований Ethernet MAC 10/100 з підтримкою MII і RMII, від 8 до 14-ти розрядний інтерфейс камери, що дозволяє зробити з'єднання з КМОП-камерами при роботі на швидкості до 67.2 Мбайт/с. Мікроконтролери STM32F407 доступні в чотирьох типах корпусів - LQFP100, LQFP144, LQFP / BGA176, і Flash-пам'яттю від 512 Кбайт до 1 Мбайта.

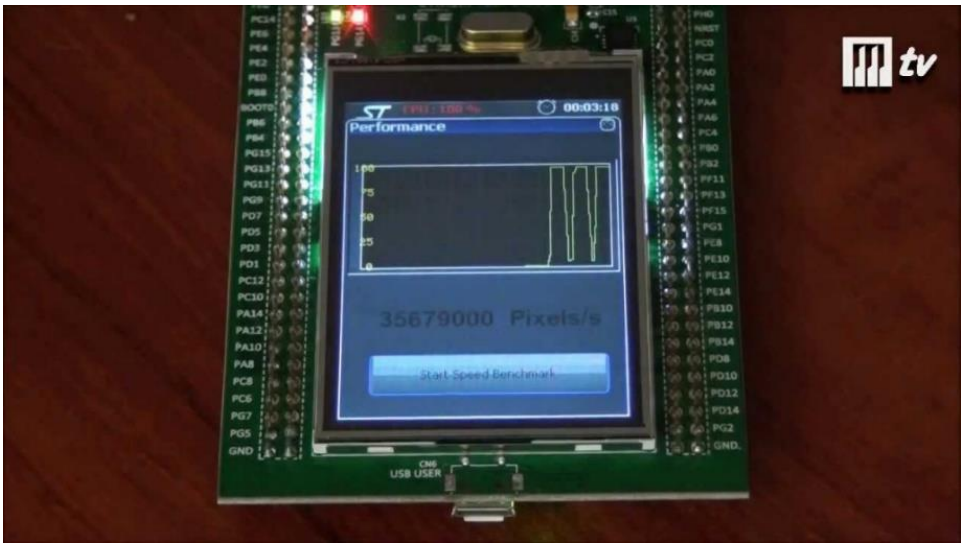


Рис. 2.2 МікроконтроллерSTM32F407VG с TFT-драйвером:

Ця лінійка контролерів створена на базі високопродуктивного ядра ARM Cortex-M4 180 Мгц і включає розширену периферію для роботи з графікою: апаратний контролер TFTLCD-дисплеїв, графічний прискорювач Chrom-ART (DMA2D) (Рис.2.2).

2. Склад серії:

- STM32F401- 84 MHz CPU / 105 DMIPS. Характерезується низьким споживанням живлення і корпусом малих розмірів, на відміну від інших класів серії STM32 F4.
- STM32F405 / 415 - 168 MHz CPU / 210 DMIPS, до 1 МБайт флеш пам'яті з розширеними можливостями шифрування.
- STM32F407 / 417 - 168 MHz CPU / 210 DMIPS до 1 МБайт флеш пам'яті з наявністю Ethernet MAC і інтерфейсом камери для STM32F405 / 415.
- STM32F427 / 437 - 168 MHz CPU / 210 DMIPS, до 2 МБайт флеш, переваги перед STM32F407 / F417 - розширені можливості шифрування.
- STM32F429 / 439 - 180 MHz CPU / 225 DMIPS, до 2 МБайт двухбанкової флеш-пам'яті з інтерфейсом SDRAM, контролером TFT LCD, технологією акселлерації Chrom-ART, послідовним аудіоінтерфейсом, забезпечує велику продуктивність і менше енергоспоживання в порівнянні з STM32F4x7 / F4x5.

3. Загальні характеристики сімейства

Структура мікроконтролера зображена на Рис.2.3, нижче приведенні основні характеристики цього сімейства пристроїв:

Блокядра (ARMCortex-M4 168 MHz):

- Ядро ARM 32-bit Cortex-M4 CPU;
- Частотатактування 168МГц, 210 DMIPS / 1.25 DMIPS / МГц (Dhrystone 2.1);
- Підтримка DSP-інструкцій;

БлокARTприскорювача (ARTAccelerator);

Блоквисокопродуктивної AHB-матрицішин (Multi-AHBbusmatrix);

Блок пам’яті (Memory):

- До 1 МБайт Flash-пам'яті;
- До 196 кбайт SRAM-пам'яті;
- Контролер SDIO (карти SD, SDIO, MMC, CE-ATA);
- FSMC-контролер (Compact Flash, SRAM, PSRAM, NOR, NAND і LCD 8080/6800);

Системний блок (System):

- Напруга живлення 1,8 ... 3,6 (POR, PDR, PVD і BOR);
- Внутрішні RC-генератори на 16МГц і 32кГц (для RTC);
- Зовнішнє джерело тактування 4 ... 26МГц і для RTC - 32,768кГц;
- Апаратне обчислення CRC, 96-бітний унікальний ID;
- Зовнішній осцилятор 32kHz +16 MHz
- Xtal осцилятори 32kHz
- Регулятор POR/PDR/PVD
- 51/82/114/140 I/Os
- Clock control

Блок роботи з аналоговими сигналами (Analog):

- Три 12-бітних АЦП на 24 вхідних каналу (швидкість до 7,2 мегасемплів, температурний датчик);
- Два 12-бітових ЦАП;

Блок DMA-контролера на 16 потоків з підтримкою пакетної передачі (16-channelDMA);

Блок управління (Control):

- 17 таймерів (16 і 32 розрядні);
- Два сторожових таймера (WDG і IWDG);

Блок комунікації (Connectivity)

- Комунікаційніінтерфейси: I2C, USART (ISO 7816, LIN, IrDA), SPI, I2S;
- CAN (2,0 B Active);
- USB 2.0 FS / HS OTG;
- 10/100 Ethernet MAC (IEEE 1588v2, MII / RMII);
- Інтерфейс цифрової камери (8/10/12/14-бітові режими);
- Апаратний генератор випадкових чисел;

Блоккриптопроцесора (Crypto/hash processor)

- Модульшифрування AES 128, 192, 256, Triple DES, HASH (MD5, SHA-1), HMAC;
- Розширений температурний діапазон -40 ... 105 ° C.

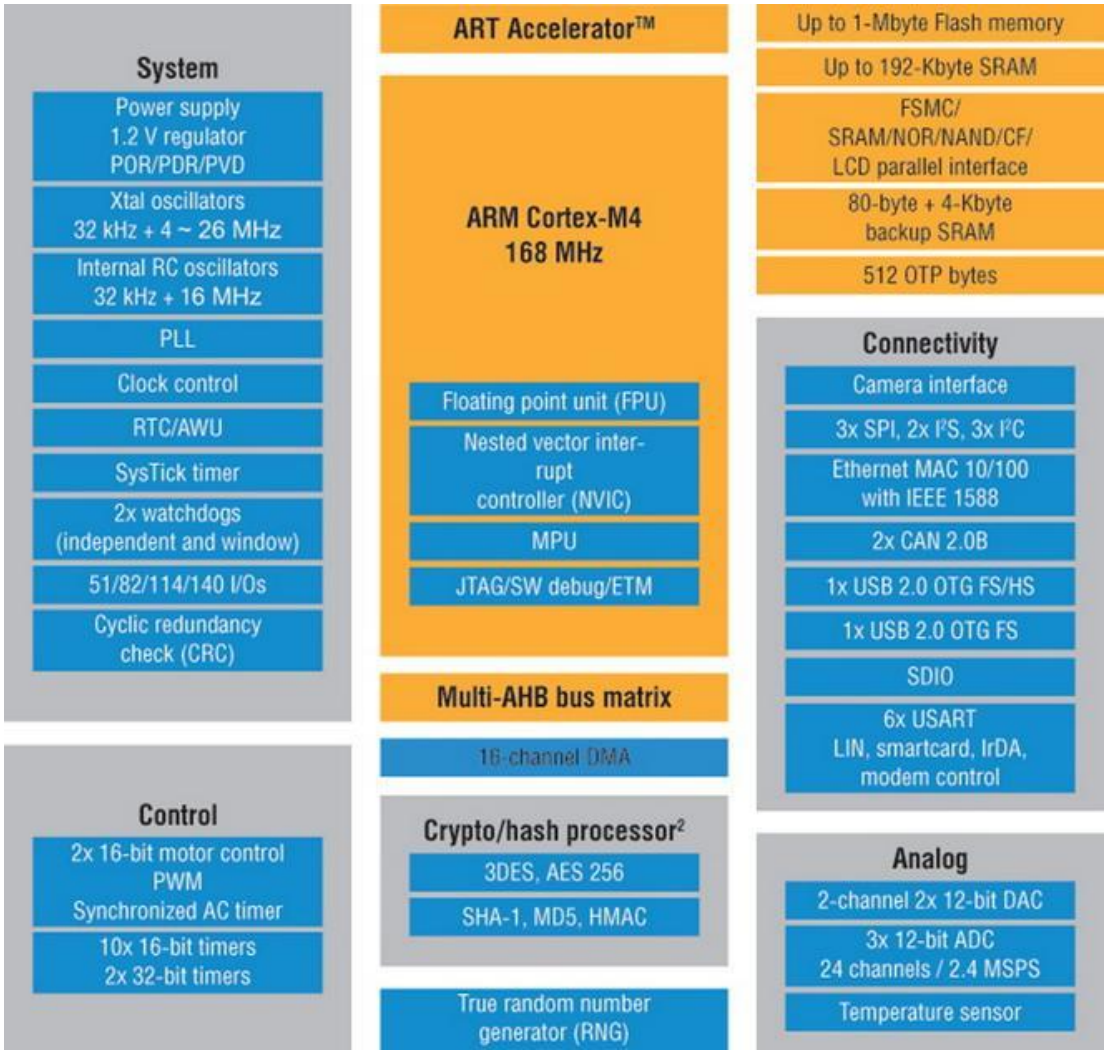


Рис. 2.3. Структура мікроконтролерів STM32F4xx

4. Вбудовані інтерфейси комунікації

Блок комунікації (Рис.2.3 Connectivity) складається із наступних інтерфейсів:

Ethernet. Даний блок присутній не в усіх продуктах сімейства, а лише в контролерах STM32F407 / STM32F417. Блок виконаний за стандартом IEEE802.3. Можлива передача даних зі швидкістю 10/100 Мбіт/с. Доступна синхронізація годин для чого протокол IEEE1588 v2 реалізований апаратно. PHY-трансивер з'єднується безпосередньо з портом MII або RMI.

USB **(Universal Serial Bus)**. Мікропроцесор має два роздільних блоку USB. Перший - USB OTG full-speed, є повністю апаратною реалізацією і сумісний зі стандартами USB 2.0, а також OTG 1.0. Працює на швидкості до 12 Мбіт / с. Підтримується робота в режимі Host / Device / OTG. Присутній SRP (Session request protocol) і HNP (Host negotiation protocol).Другий - USB OTG high-speed працюєврежимі Host / Device / OTG звисокоюшвидкістю480 Мбіт / с, для чого необхідний блок передатчика, що працює на високій швидкості через спеціальний ULPI-інтерфейс.

SDIO **(Secure Digital Input/Output)**. Інтерфейс дозволяє працювати з картами SD / SDIO / MMC-картами пам'ятіі дисковими контролерами CE-ATA. У восьми бітному режимі несуча частот аобміну даними складає 48 MHz. Контролер відповідає такимс тандартам: SD Memory Card 2.0, MultiMediaCard System 4.2 (робота в режимах 1/4/8 біт), SD I / O Card 2.0 (режими 1гоі 4хбіт), CE-ATA 1.1.

SPI **(Serial Peripheral Interface)**. Пристрій включає три блоки SPI, кожен зяких працює в режимі Master (Multimaster) або в режимі Slave, передаючи дані у напів-дуплексному, дуплексному або сімплексному режимі. Підтримується апаратний розрахунок контрольних сум CRC дл япідвищення завадостійкості каналу зв'язку: так CRC може бути переданий останнім байтом слова в режимі

Tx, присутня автоперевірка правильності CRC останньо гобайта. Блок пристрої SPI1 працює нашвидкостях аж до 37,5 Мбіт / с. Інші обмежені максимальною швидкістю в 21 Мбіт / с.

USART **(Universal Synchronous Asynchronous Receiver Transmitter)**. У мікроконтролер вбудовано чотири блоки USART ідва UART. Блоки USART1 і USART6 допускають високошвидкісний обмін даними н ашвидкості до 10,5 Мбіт / с. Інші підтримують швидкість не більше 5,25 Мбіт / с. В будована підтримка передачі даних згідно стандарту NRZ (Non Return to Zero).

Обмін даними здійснюється з використанням 8- або 9-бітних блоків, один або два біти яких виділені як стоп-біти і біти перевірки парності. USART можна конфігурувати на режим SPI, блок USART при цьому виступає в ролі ведучого пристрою SPI. Використовуючи блок USART можна організувати підключення до інтерфейсу LIN, яка знайшла застосування в автомобільній промисловості, або налаштувати на енкодинг / декодинг ІЧ-сигналу IrDA. Можлива робота з модемами по лініях управління RTS і CTS. Підтримується робота зі смарт-картками.

I2C **(Inter-Integrated Circuit)**. На борту МК містить три блоки I2C, що підтримують роботу в режимі Master / Slave (ведучий або ведений), а також у режимі Multimaster (режим в якому на шині присутні кілька Master-пристроїв, які поділяють спільні ресурси Slave, або по черзі змінюють свій стан з Master на Slave і назад). У складі пристрою є модуль діагностики та виправлення пакетних помилок PEC. Використовується 7-бітний і 10-бітний режим адресації. Підтримуються загальноприйняті для протоколу швидкості обміну даними до 100 kHz в простому режимі і 400 kHz в режимі швидкого обміну даними. Модулі можуть бути сконфігурірованни на розширені протоколи SMBus 2.0 і PMBus.

I2S **(Inter-Integrated Sound)**. У мікроконтролері присутні два мультиплексованих блоку I2S з вбудованим SPI. Обидва модулі можуть бути сконфігурованими на роботу в режимі Master або Slave. Дані передаються по 16, 24 або 32 біта дуплексно або сімплексно.

Серед підтримуваних протоколів такі: Phillips I2S, PCM, MSB і LSB з вирівнюванням даних. Інтерфейс I2S був розроблений для обміну звуковими даними в цифровому форматі. Відтепер для тактування присутній окремий PLL, який робить можливим генерацію частот аудіосемплів від 8 до 192 kHz з похибкою не більше 0,01%.

CAN **(Controller Area Network)**. На борту знаходиться два CAN-модуля, що працюють за стандартами 2.0A і 2.0B, швидкість роботи при цьому досягає 1 Мбіт / с. Модулі можуть працювати зі стандартними, а також з розширеними кадрами. Модуль CAN містить три буфера передачі, трьохкаскадний FIFO-стек і 28 банків фільтрів повідомлень (розподілені і масштабуються).

DCMI **(Digital Camera Interface)**. Присутній в контролерах STM32F407 і STM32F417. За допомогою даного інтерфейсу можна організувати пряме підключення до камери або CMOS-матриці. Можлива внутрішня і зовнішня синхронізація, робота в безперервному режимі, автообрізка зайвих частин зображення. Серед підтримуваних форматів 8/10/12/14-бітове прогресивне відео, YCbCr 4: 2: 2 і RGB 565, JPEG.

FSMC (Flexible Static Memory Controller). Блок використовується для підключення рідкокристалічних дисплеїв якої зовнішньої пам'яті безпосередньо. Блок міститься лише в 100-, 144- або 176-пінових корпусах.

Можливе сполучення зпідключеною синхронної або асинхронної пам'яттю або PCMCIA- пристроями. В основному блок призначений для видачі даних МК у відповідному підключеним пристроям вигляді без зайвих витрат процесорного часу на перекодування даних.

Таким чином кожнтй зовнішній пристрій має власний адрес пулу, власні сигнали для управління. Подавши необхідний сигнал вибору мікросхеми можна отримати доступ до того чи іншого зовнішнього пристрою (одночасне використання не припустимо). Можливе підключення таких типів пам'яті як NAND Flash, Compact Flash, NOR Flash, SRAM і PSRAM. Інтерфейс налаштований для роботи з LCD-контролерами Motorola 6800 і Intel 8080, однак

може бути легко використаний для сполучення з контролерами інших виробників.

5. Робота з аналоговими сигналами

Блок роботи з аналоговими сигналами (Рис.2.3 Analog)містить три АЦП і два одноканальних ЦАП. АЦП має хорошу роздільну здатність 12 біт і високу швидкість перетворення - 2,4 МСемпла в звичайному режимі і 7,2 МСемпла - в потрійному режимі. Максимально доступне число аналогових каналів - 24. Як і в більшості сучасних МК, присутній генератор опорного напруги. Гнучка система налаштувань вбудованого аналогового мультиплексора дозволяє задавати будь послідовності перетворення аналогових каналів (за винятком одночасного перетворення одного каналу на декількох АЦП). Налаштування АЦП дозволяють виробляти одноразові і циклічні вимірювання. Для проведення перетворення на максимальних швидкостях необхідно дотримуватися діапазон напруги живлення 2,4 ... 3,6 В. При зниженні напруги до 1,8 (1,7) В швидкість перетворення знижується до 1,2 мегасемплов. Для контролю внутрішньої температури мікроконтролера вбудований температурний датчик. На його виході формується напруга в залежності від навколишньої температури. Вихід датчика через мультиплексор підключається до АЦП. Використовуючи температурний датчик, можна вимірювати температуру від -40 до 125 ° С з точністю ± 1,5 ° С.

ЦАП має роздільну здатність 12 біт, перетворення можливо в 8/12-бітовому форматі з вирівнюванням цього результату по лівому або правому краях. Так як ЦАП містить два канали, тобто можливість формування стереосигнала. Доступна функція автоматичної генерації шумового сигналу з мінливою амплітудою або трикутного сигналу.

6. Блок-схема периферії

На рис. 2.4 зображена периферія МК STM32F407VGT6, що використовується в рекомендованій виробником платі для швидкого старту STMF4-DISCOVERY

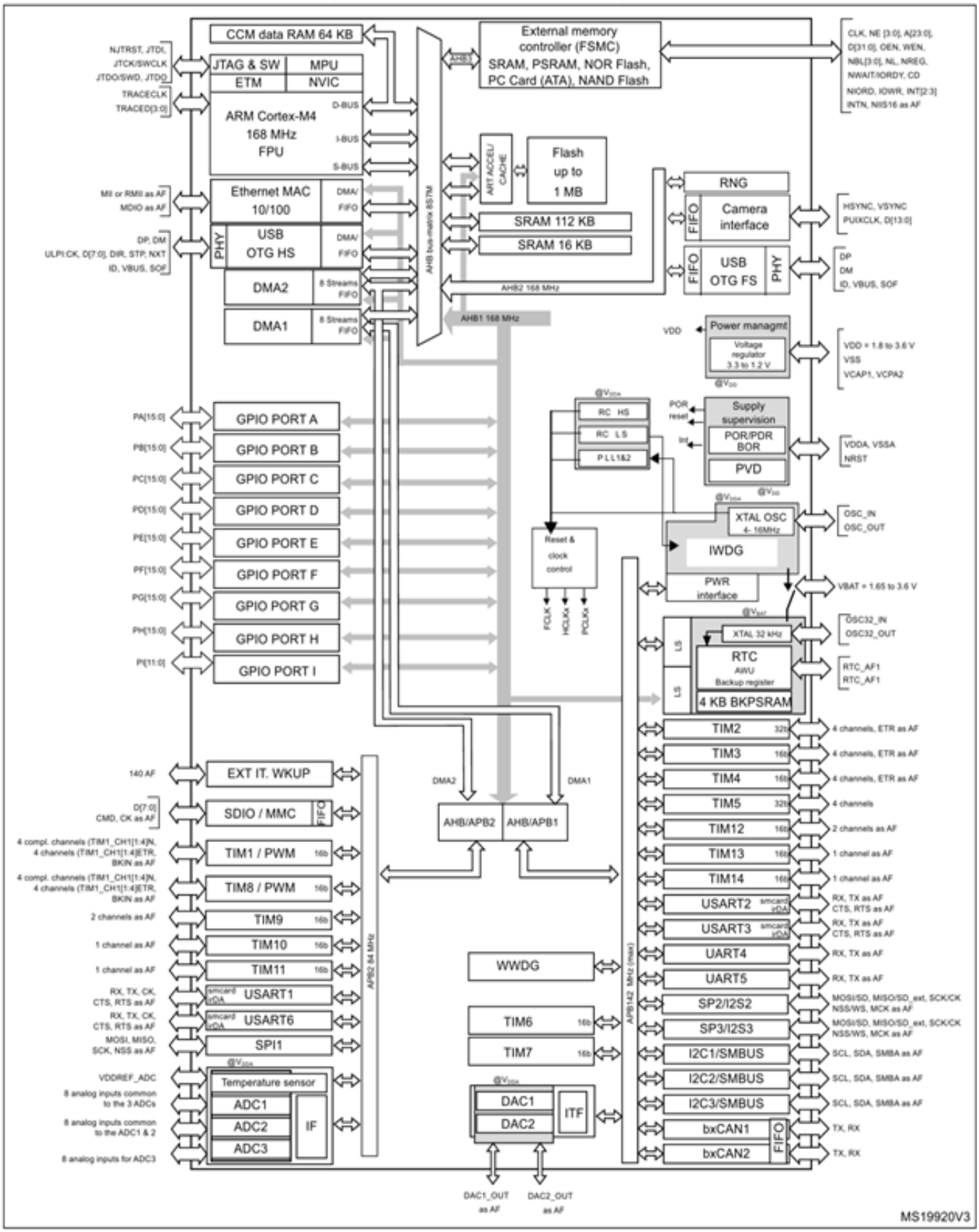


Рис. 2.4. Периферія STM32F405xx, STM32F407xx

Тема 3.3 Периферія мікроконтролера STM32F407VG. Організація пам’яті, шин даних, тактування та живлення.

1. Пам’ять та архітектура шини даних

Сучасні процесори містять безліч різних шин, до яких підключаються інші пристрої системи. На зорі комп'ютерної епохи шиною (bus) називали просто пачку провідників, що з'єднують декілька пристроїв з процесором, який міг одночасно працювати лише з одним пристроєм, а інші в цей час простоювали, оскільки шина була на всіх одна. До того ж, всі пристрої були змушені працювати на одній швидкості (самого повільного пристрою), що утримувало зростання продуктивності систем.

Сьогодні шини стали складніше, їх число збільшилося на порядок. Роздільні шини дають можливість працювати з декількома пристроями одночасно, причому на різних швидкостях: у кожної шини може бути своя швидкість. Можна гнучко управляти енергоспоживанням, відключаючи невикористовувані пристрої й цілі шини. Для управління всією цією кількістю шин знадобилося вводити спеціальний контролер, керуючий обміном даними між шинами і процесором.

Шинна матриця - це розвиток ідеї простого контролера шини: тут шини з'єднані так, що пристрої можуть взаємодіяти безпосередньо, не через ядро. Також вона керує доступом до не-вирівняних даних (адреси яких не кратні 4, як прийнято в 32-бітових архитектурах) і атомарним доступом до окремих бітів у спеціально виділеному діапазоні (технологія bit-banding)

В мікропрцесорі Cortex-M3 застосовано 4 шини, підключених до матриці:

- Шина ICode (вибірки інструкцій і векторів переривань) використовується для користувацького коду. 32-бітна шина AHB (Advanced High - performance Bus)-

Lite типу.

- Шина DCode (вибірки / запису даних і відладочного доступу) використовується для користувацького коду. 32-бітна шина AHB-Lite типу.
- Шина System (вибірки інструкцій і векторів переривань, а також вибірки / запису даних і відладочного доступу в системному просторі) використовується для внутрішніх компонентів МК. 32-бітна шина AHB-типу.
- Шина PPB (Private Peripheral Bus) (вибірки / запису даних і відладочного доступу) використовується для периферії. 32-бітна шина APB (Advanced Peripheral Bus) - типу.

Advanced High - performance Bus (AHB) - це відносно нова специфікація для більш продуктивних шин, шини цього типу використовують, в основному, для зв'язку високошвидкісних внутрішніх компонентів, а шину APB, як більш повільну - для периферії типу GPIO, UART та ін.

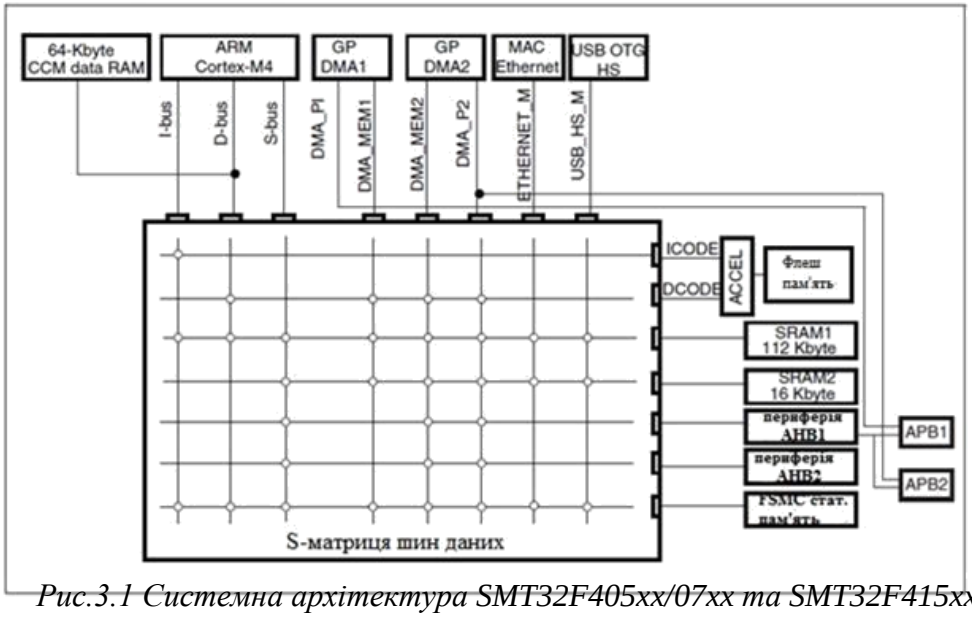


Рис.3.1 Системна архітектура SMT32F405xx/07xx та SMT32F415xx/07xx

Архітектура системи:

У мікропроцесорах STM32F405xx / 07xx та STM32F415xx / 17xx, основна система являє собою 32-бітову багатошарову AHB матрицю шин, яка з'єднує вісім мастер - шин та сім слейв - шин.

Наступні майстер - шини:

- Шина ядра, I-шина, D-шина та S-шина;
- шина пам’яті DMA-1;
- шина пам’яті DMA-2;

- периферійна шина DMA-2;
- Ethernet - шина DMA;
- USB OTG HS DMA шина;

Слейв - шини складаються з:

- ICode-шини вбудованої флеш-пам’яті;
- DCode-шини вбудованої флеш-пам’яті;
- шини головної вбудованої SRAM1 (112kB);
- шини допоміжної вбудованої SRAM2 (16kB);
- шини AHB1 периферії;
- шини AHB2 периферії;
- шини FSMC.

Організація пам’яті у мікроконтролерах STM32F407VG відбувається за наступними правилами:

1) Програмна пам'ять, пам'ять даних, регістри і порти введення / виводу організовані в межах лінійних 4 Гбайт адресного простору.

2) Байти кодуються в пам'яті в прямий порядок. Найменшим номером байта в слові вважається молодший байт слова, найбільший номер отримує найзначніший байт слова.

3) Адресний простір пам'яті ділиться на 8 основних блоків, кожен з 512 Мб.

4) Всі області пам'яті, які не виділені на чіпі пам'яті і периферійних пристроях вважаються "захищеними".

На рис. 3.2 представлена карта секторів пам’яті для мікроконтролерів STM32F40xx.

Рис.3.2 Карта секторі пам’яті мікроконтролера SMT32F40xx

2. Вбудований статичний ОЗП

У STM32F405xx / 07xx і STM32F415xx / 17xx заплановано 4 Кбайт резервного статичного ОЗП та 192 Кбайт системного ОЗП.

Вбудований статичний ОЗП має доступ як до байтів, півслів (16 біт) або повних слів

(32 біта).

Операції читання і запису виконуються на швидкості процесора з нульовим очікуванням.

Статичний ОЗП ділиться на три блоки:

1) SRAM1 і SRAM2 відображається в адресі 0x2000 0000 і доступна для всіх AHB майстрів.

2) SRAM3 (доступна на STM32F42xxx і STM32F43xxx) відображається за адресою 0x2002 0000 і доступна для всіх майстрів AHB.

3) CCM (ядро, поєднане з пам'яттю) відображається за адресою 0x1000 0000 і доступно лише за допомогою CPU через D-Bus.

3. Флеш-пам'ять

Інтерфейс флеш-пам'яті здійснює доступ до флеш пам'яті через AHB - шини I-Code і D-Code. Він реалізує стирання і програмні операції і механізми захисту читання / запису. Це прискорює виконання коду з системою попередньої вибірки і кешування лінії.

Флеш-пам'ять організована таким чином (рис. 3.3):

- Основний блок пам'яті розділений на сектори
- Системна пам'ять, з якої пристрій завантажується в режимі завантаження пам'яті системи
- 512 OTP (одноразово програмованих) байт для даних користувача.

- Опціональні байти для налаштування захисту читання і запису, сторожового таймера і скидання, коли пристрій знаходиться в режимі очікування або зупинки.

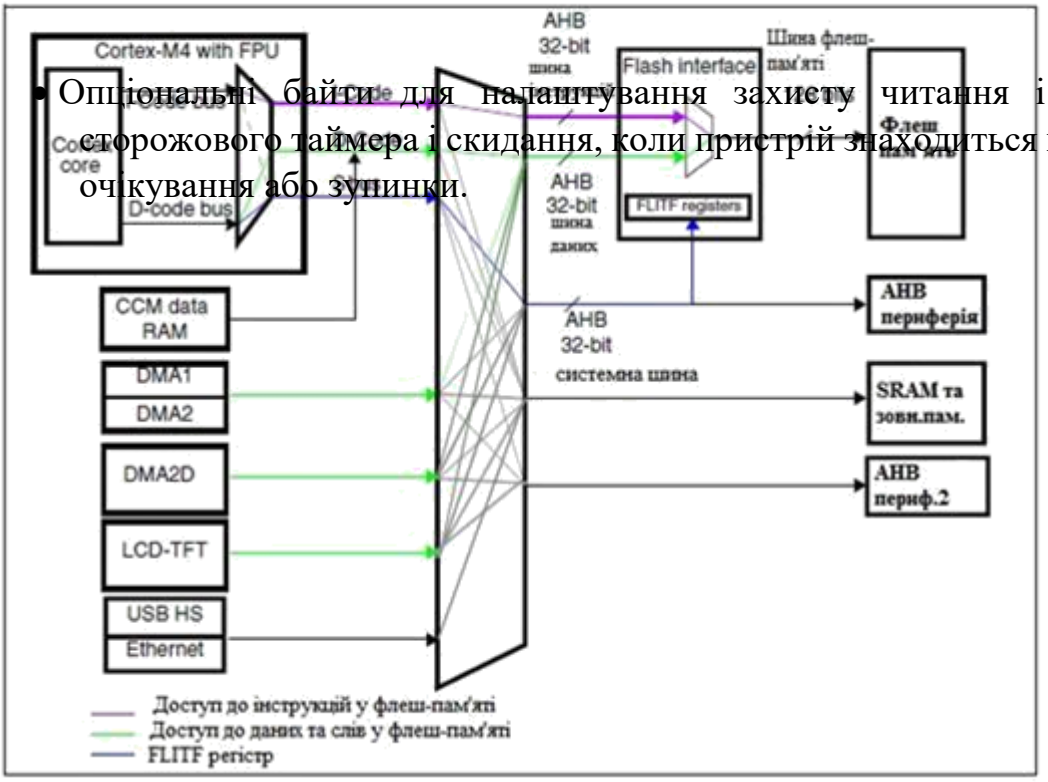


Рис. 3.3 Організація цин для доступу до флеш-пам'яті мікроконтролера SMT32F40xx

4. Конфігурація завантаження.

У STM32F4xx, три різних режими завантаження можна вибрати через контакти BOOT [1: 0]:

Піни вибору режиму BOOT		Режими BOOT	Результат
BOOT1	BOOT0		
x	0	Головна флеш-пам'ять	Головна флеш-пам'ять обрана як завантаж. простір
0	1	Системна пам'ять	Системна пам'ять обрана як завантаж. простір
1	1	Вбудований SRAM	Вбудований SRAM обраний як завантаж. простір

Значення по контактам BOOT фіксуються на 4-му передньому фронті імпульсів SYSCLK після скидання. Залежить від користувача, як

встановити прапорці boot1 і boot0 після скидання, щоб вибрати потрібний режим завантаження.

Boot0 є окремим піном, а boot1 сумісний з GPIO піном. Після того як boot1 був просемпльований, відповідний GPIO контакт є вільним і може бути використаний для інших цілей. Виходи BOOT також перезаписуються, коли пристрій виходить з режиму очікування. Отже, вони повинні бути збережені в необхідній конфігурації режиму завантаження, коли пристрій знаходиться в режимі очікування. Після того як затримка запуску закінчилася, процесор отримує верхнє стекове значення з адреси0x0000 0000, потім починає виконання коду з пам'яті початкового завантаження, починаючи з 0x0000 0004.

Джерела живлення

Мікропроцесор вимагає напруги живлення від 1,8 до 3,6 В. Вбудований лінійний регулятор напруги використовується для живлення 1.2 V. Годинник реального часу (RTC), регістри RTC резервного, і резервного статичного ОЗП (БКП статичного ОЗП) можуть живитися від напруги VBAT, коли основна напруга живлення VDD вимкнена.

На рис. 3.4 зображено:VBAT - резервна напруга живлення мікропроцесора;VDD - основна напруга живлення (від 1.8 В до 3.6 В); ADC - аналогово - цифровий перетворювач.

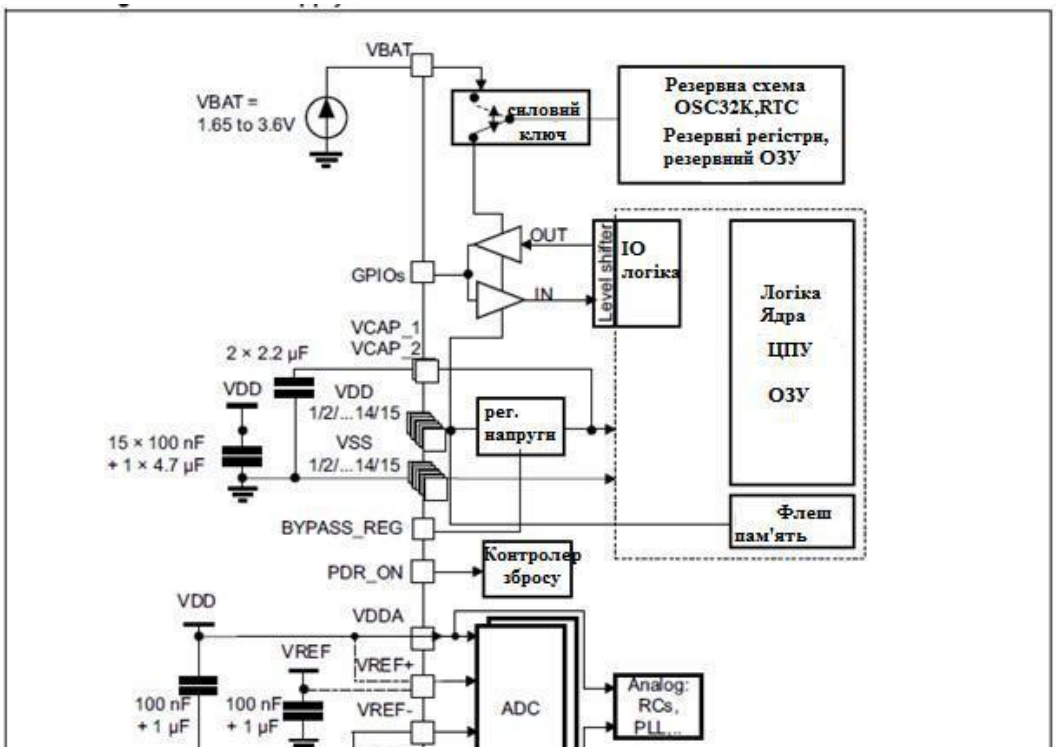


Рис.3.4 Система живлення мікроконтролера SMT32F40xx

5. Тактування мікроконтролера STM32F40xx.

Згідно документації (Reference manual RM0041) в якості джерела тактових сигналів (SYSCLK - системні тактові сигнали) можуть виступати три джерела:

- HSI oscillator clock - внутрішній високочастотний генератор
- HSE oscillator clock - зовнішній високочастотний генератор
- PLL clock - ситема ФАПЧ

HSE oscillator clock -зовнішній високочастотний генератор

Джерелом сигналів для HSE генератора може бути як зовнішній тактовий генератор, так і звичайний кварцовий або керамічний резонатор.

Для встановленого на платі STM32-Discovery мікроконтролера частота зовнішнього сигналу не повинна перевищувати 24 МГц, при використанні зовнішнього тактового генератора, а при використанні кварцового (керамічного) резонатора його частота повинна бути від 4 до 24 МГц.

Зовнішній сигнал може мати форму пили, синусоїди або прямокутних імпульсів зі скважністю 50%.

HSI oscillator clock -внутрішній високочастотний генератор

HSI генератор являє собою RC-генератор з частотою 8 МГц, він тактує інтерфейс програмування флеш-пам'яті, може бути джерелом тактових сигналів (SYSCLK), а так само може служити джерелом опорних сигналів для ФАПЧ (однак при цьому його частота ділитися на два, т .е. становить 4 МГц).

Даний генератор проходить калібрування на заводі і виробник гарантує точність в 1% при температурі 25 градусів Цельсія.

PLL clock -ситема ФАПЧ

Система ФАПЧ (фазове автопідстроювання частоти) виробляє множення опорного сигналу (доступні коефіцієнти від 2 до 16), однак частота на виході системи ФАПЧ повинна лежати в межах 16-24 МГц.

Так само є два вторинних джерела тактових сигналів:

LSI RC - внутрішній низькочастотний RC-генератор (40 кГц)

LSE crystal - зовнішній низькочастотний кварцовий генератор (32 кГц)

Від внутрішнього низькочастотного RC-генератора тактується сторожовий таймер, так само від нього може тактіроваться RTC-таймер, за сигналами якого можна виводити мікроконтролер із сплячого режиму.Всі ці джерела можна незалежно вмикати і вимикати, це особливо актуально коли необхідно знизити споживану потужність.

Тема 3.4 Порти GPIO та їх характеристики

Безпосереднє управління станом пінів МК здійснюється за допомогою портів GPIO. В МК STM32 може бути до 11 незалежних 16-розрядних портів GPIO (G*eneral purpose input output)* , що позначаються літерами від A до K.

Порти можуть роботи в наступних режимах

- Input (вхід): Floating (плаваючий вхід з високим вхідним опором, нічим не навантажений), Pull-up (з верхнім навантажувальним резистором), Pull-down (з нижнім навантажувальним резистором)
- Output (вихід): Push-Pull двотактний ключ (Pull-up з верхньої навантаженням, Pull-down з нижнім навантаженням або no Pull просто двотактний вихід), Open Drain відкритий стік (Pull-up, Pull-down або no Pull). У режимі виходу може бути запрограмована швидкість роботи порту: 2 МГц, 25 МГц, 50 МГц або 100 МГц.
- Alternate Function (альтернативнафункція): Push-Pull (Pull-up, Pull-down або no Pull), Open Drain (Pull-up, Pull-down або no Pull)
- Analog (аналогова лінія): цей режим необхідний, коли вивод використовується як канал ADC (АЦП) або вихід DAC (ЦАП).

Таким чином, при роботі на вхід можливі режими:

Вхід – Hi-Z (*Input floating)*

Вхід – підтяжка вгору (*Input pull-up)*

Вхід – підтяжкавниз (*Input-pull-down)*

Вхід – аналоговий (*Analog)*

При роботі порту на вихід :

- Вихід – з відкритим колектором (*Output open-drain)*
- Вихід – двотактний (*Output push-pull)*

- Альтернативніфункції – вихідтипу «звідкритимколектором» (*Alternatefunction push-pull)*
- Альтернативніфункції– двотактнийвихід (*Alternate function open-drain*)

Схемотехніка однієї лінії порта наведена на рис.4.1

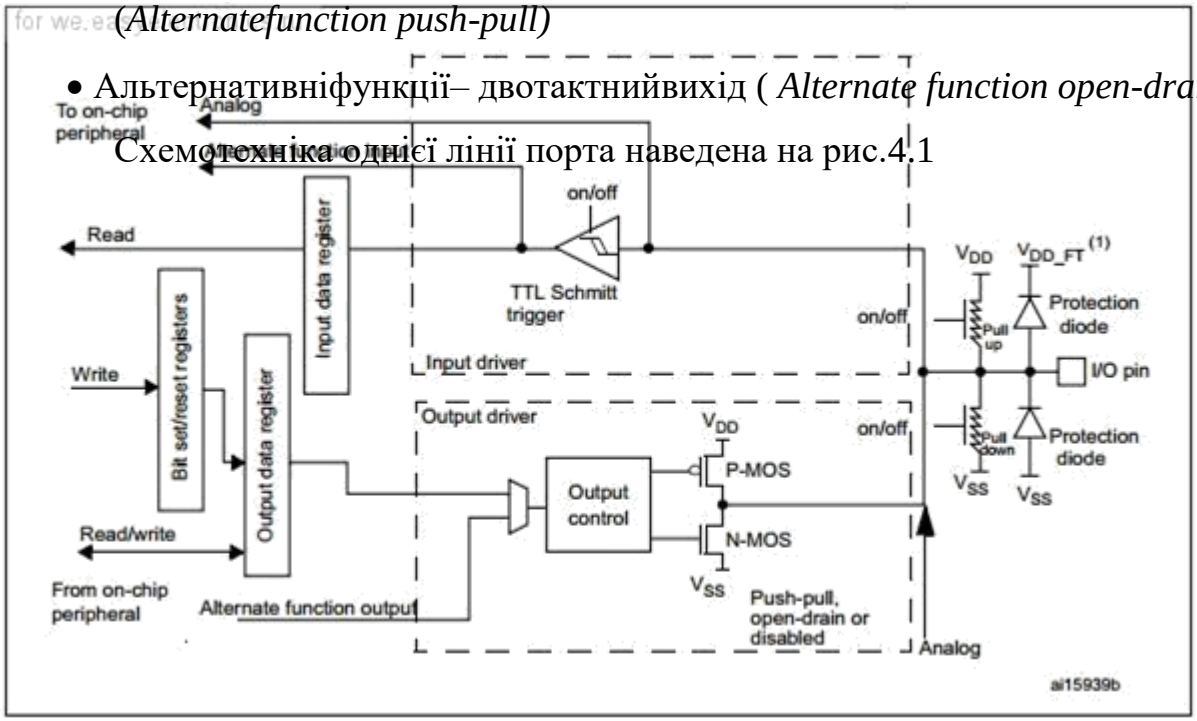


Рис. 4.1 Схемотехніка однієї лінії порта

В режимі входу сигнал з піна I/Opin через триггер Шмітта поступає або на вхідний регістр (*Inputdataregister*), або на вхід альтернативної функції *(Alternatefunctioninput)*, або як аналоговий вхід *(Analog).* В режимі виходу дані з вихідного регістру (*Outputdataregister*) або альтернативна функція *(Alternatefunctionoutput)* поступає на вихідний каскад на компліментарній парі трашзисторів, який може працювати в якості двотактного виходу або виходу з верхнім або нижнім навантаженням. На рис. 4.1 показано також керування верхнім та нижнім навантажувальними резисторами

Під час активного сигналу скидання і відразу після його закінчення всі альтернативні функції на виводах портів не активні (за винятком виводів,

задіяних під JTAG), і всі порти виявляються сконфігурованими як плаваючі входи з високим опором.

Виводи генератора LSE (OSC32_IN і OSC32_OUT) можуть використовуватися як GPIO (PC14 і PC15 відповідно), якщо генератор LSE вимкнений. LSE має пріоритет перед функцією GPIO.

Виводи генератора HSE (OSC_IN і OSC_OUT) можуть використовуватися як GPIO (PH0 і PH1), коли генератор HSE вимкнений (основний кварцовий генератор не використовується, що буває дуже рідко). HSE має пріоритет над функцією GPIO.

Для керування цим блоком існує 12 32х бітних регістрів:

1) GPIO port mode register (GPIOx_MODER) **керування режимами**
2) GPIO port output type register (GPIOx_OTYPER) **регістр задання режиму виходу**
3) GPIO port output speed register (GPIOx_OSPEEDR) **керування швидкістю вивода**
4) GPIO port pull-up/pull-down register (GPIOx_PUPDR) **керування***pull-up/pull-down*
5) GPIO port input data register (GPIOx _IDR) **вхідні дані**
6) GPIO port outp ut data register (GPIOx_ODR) **вихідні дані**
7) GPIO port bit set/reset register (GPIOx _BSRR) **керування встановленням/ скиданням бітів**
8) GPIO port configuration lock register (GPIOx_LCKR) **керування блокуванням конфігурації**
9) GPIO alternate function low register (GPI Ox_AFRL) **керування альтернативними функціями**

10. GPIO alternate function high register (GPIOx_AFRH) **керування альтернативними функціями**

11. GPIO Port bit reset register (GPIOx_BRR) **керування скиданням бітів вихідного регістру даних**

1. *GPIO port mode register (GPIOx_MODER)* – **регістр режимів**

Цей регістр відповідає за налаштування пінів у різні режими, такі як: вхід, вихід, аналоговий вхід або альтернативна функція.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Біти [2n:2n+1] відповідають за: 00: Вхід(за замовчуванням) 01: Вихід Альтернативна функці/Аналоговий режим

2. *GPIO port output type register (GPIOx_OTYPER*) – **регістр завдання виходу**

Цей регістр відповідає за тип виходу. Встановлення нуля або одиниці вмикає відповідний режим push-pull чи open-drain. За замовчуванням режим встановлений у push-pull.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Біти 31:16 зарезервовані Біти 15:0 OT конфігураційні біти (y= 0…15)

3. GPIO port output speed register (GPIOx_OSPEEDR) – **керування швидкістю вивода**

Address offset: 0x08

Reset values:

- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

4. *GPIO port pull-up/pull-down register (GPIOx_PUPDR*) - **керування***pull-up/pull-down*

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

5. *GPIO port input data register (GPIOx_IDR)* – **вхідні дані**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDR[15:0]:** Port input data
These bits are read-only. They contain the input value of the corresponding I/O port.

6. *GPIO port outp data register (GPIOx_ODR)* **вихідні дані**

Містить дані для вивода

Наприклад запис в вихідний порт C GPIOC->ODR = 0xF0FE

змінить його стан на 0xF0FE (0b1111000011111110).

Можлива зміна одного біта. Так, щоб встановити контакт PC8 незалежно від всіх інших контактів

GPIOC :

GPIOC->ODR |= 0x00000100; //

(0b00000000000000000000000100000000)

Щоб скинути контакт PC8 незалежно від всіх інших контактів на GPIOC :

GPIOC->ODR &= ~(0x00000100); //

(0b00000000000000000000000100000000) Де OR (|) or AND(&)

Однак швидче (за один такт) можнв змінити лише один біт за допомогою наступного регістра

7. *GPIO port bit set/reset register (GPIOx_BSRR)***керування встановленням/скиданням бітів**

Цей регістр відповідає за встановлення або скидання біту на певному піні. В нього можливо лише записувати значення, зчитування не є можливим.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Біти 31:16 Можуть бути записані словом, половиною слова або побітно. Читання повертає значення 0x0000

- Нічого не відбувається
- Скидання у 0 відповідного біту

Біти 15:0 Можуть бути записані словом, половиною слова або побітно. Читання повертає значення 0x0000

0: нічого не відбувається

1: Встановлення в 1 відповідного біту

Приклад встановлення/ скидання бітів

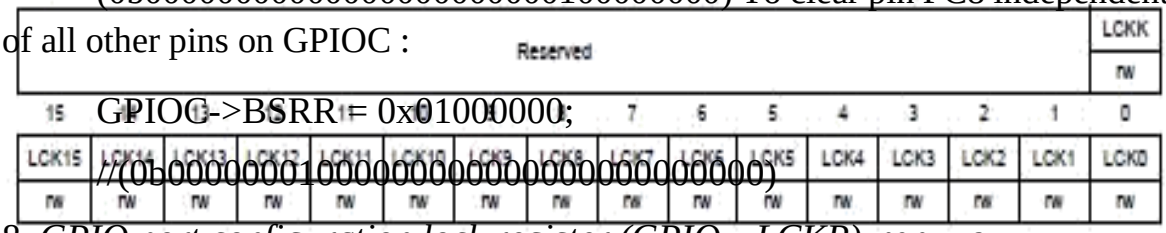
To set PC8 independent of all other pins on GPIOC :

GPIOC->BSRR = 0x00000100;//

(0b00000000000000000000000100000000) To clear pin PC8 independent of all other pins on GPIOC :

GPIOC->BSRR = 0x01000000;

//(0b00000001000000000000000000000000)



8. *GPIO port configuration lock register (GPIOx_LCKR)* **керування блокуванням конфігурації**

Для унеможливлення зміни налаштувань порту в мікроконтролерах STM32 використовується регістр GPIOx_LCKR. Його молодші 15 біт відповідають за відповідні лінії порту введення / виводу. Біт 16, встановлений в 1, дозволяє блокування зміни налаштувань. Всі біти доступні на читання / запис.

Bit 16 LCKK[16]: Lock key

- 0: Port configuration lock key not active
- 1: Port configuration lock key

active Bits 15:0 LCKy: lock bit y

- 0: Port configuration not

locked 1: Port configuration locked

Алгоритм установки захисту виглядає наступним чином:

- Встановити біт 16 GPIOx_LCKR.
- Скинути біт 16 GPIOx_LCKR.
- Встановити біт 16 GPIOx_LCKR.
- Прочитати GPIOx_LCKR
- Повторно прочитати GPIOx_LCKR

Приклад: заблокувати біти [15: 0], відповідної конфігурації

code-block:: c

WR LCKR[16] = ‘1’ + LCKR[15:0]
WR LCKR[16] = ‘0’ + LCKR[15:0] WR
LCKR[16] = ‘1’ + LCKR[15:0]

RD LCKR

RD LCKR[16] = ‘1’ (optional, confirm only)

9. *GPIO alternate function low register (GPIOx_AFRL)* Регістр призначає альтернативну функцію ніжкам портів з номерами біт від 7 до 0. Наприклад, четвірка бітів AFRL2 врегістрі GPIOB_AFRL призначить ніжці 2 порту GPIOB (PB2) будь-яку альтернативну функцію від AF1 = 0 до AF15 = 15. Зсув адреси для GPIOx_AFRL рівно 0x20, значення після скидання 0x00000000 (після скидання всі ніжки портів виконують альтернативну функцію AF0).

10. GPIO alternate function high register (GPIOx_AFRH)

Регістр призначає альтернативну функцію ніжкам портів з номерами бітів від 15 до 8. Наприклад, четвірка біт AFRH13 в регістрі GPIOA_AFRH призначить ніжці 13 порту GPIOA (PA13) будь-яку альтернативну функцію від AF1 = 0 до AF15 = 15. Зсув адреси для GPIOx_AFRH дорівнює 0x24, значення після скидання 0x00000000 (після скидання всі ніжки портів виконують альтернативну функцію AF0).

11. *Port bit reset register (GPIOx_BRR)* **керування скиданням бітів**

Даний регістр виробляє скидання високого рівня лінії, встановленої в регістрі GPIOx_ODR. Задіяні тільки молодші 16 біт, доступних тільки для запису

12. *RCC AHB1 peripheral clock enable register (RCC_AHB1ENR*)

Це регістр який відповідає за роботу не тільки портів але й іншої периферії. Після ввімкнення МК необхідно програмно ввімкнути і відповідні порти у цьому регістрі. Наприклад для порту С це буде 3 біт.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHS ULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved		DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	Reserved	
	rw	rw	rw	rw	rw	rw			rw	rw			rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved			GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Щоб порт GPIOx почав працювати, і можна було програмно керувати станом його ніжок, потрібно включити тактування порту, для чого в розряд GPIOxEN регістра RCC AHB1 треба записати одиницю. Для цієї мети служать зручні бібліотечні функції (у цьому прикладі дозволено тактування порту GPIOA):

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

GPIO alternate function low register (GPI Ox_AFRL) **керування альтернативними функціями**

GPIO alternate function high register (GPIOx_AFRH) **керування альтернативними функціями**

11. *Port bit reset register (GPIOx_BRR)* **керування скиданням бітів**

Даний регістр виробляє скидання високого рівня лінії, встановленої в регістрі GPIOx_ODR. Задіяні тільки молодші 16 біт, доступних тільки для запису

Приклад використання портів:

Для порту E:

Задати 12 пін в режим входа з підтяжкою вгору

Задати 13 пін в режим виходу, а також виставити режими open-drain, pull-up

Існує декілька способів налаштування входів, а саме:

- За допомогою встановлення конкретних бітів.
- За допомогою написання HEX значення регістра.
- За допомогою макросів.

В прикладі застосовано перші два способи. Алгоритм налаштування портів:

- Увімкнення тактування потрібного виводу (RCC->AHB1ENR регістр).
- Встановлення направленості виводу (GPIOx->MODER регістр).
- Встановлення типу вихідного стану вивода (GPIOx->OTYPER регістр)
- Встановлення типу підтяжки (GPIOx->PUPDR регістр).
- Встановлення швидкості виводів (GPIOx->OSPEED регістр).

Для простого виводу (наприклад, висвітлення діоду) достатньо лише виконати перший пункт та встановити логічну одиницю на виході. Програма, що реалізує умови прокладу, наступна:

```
#include "stm32f4xx.h"

int main(void)
{
SystemInit(); //Конфігурація частот процесова

//Варіант №1 з використанням назв регістрів
if(RCC_AHB1ENR_GPIOEEN != 1)//Якщо тактування відсутнє
        RCC->AHB1ENR |= RCC_AHB1ENR_GPIOEEN; //Встановити тактування
```

```
                              (GPIO_MODER_MOD      GPIO_MODER_MODER
      GPIOE->MODER =~   ER12 |               13);
      //Скидання
бітів
                              GPIO_MODER_MODE      //Встановленян
      GPIOE->MODER =    R13_0;               на вихід
      GPIOE-                  GPIO_OTYPER_OT_13; //Відкритий
>OTYPER            =    колектор
                           &=~
      GPIOE->PUPDR GPIO_PUPDR_PUPDR13;
                              GPIO_PUPDR_PUPD      //З підтяжкою на
      GPIOE->PUPDR =    R13_0;               джерело

      //Варіант №2 є HEX варіант
      //RCC->AHB1ENR = 0x00000010;
//GPIOE->MODER = 0x04000000; //GPIOE-
>OTYPER = 0x00002000; //GPIOE->PUPDR =
0x04000000; //GPIOE->OSPEEDR =
0x08000000;

      while(1)
      {
      GPIOE->BSRRL |= GPIO_BSRR_BS_13;
//Одиницянавиході//GPIOE->BSRRL = 0x2000;

      } }
```

Тема 3.5 Таймери

Мікроконтролер STM32 має у своєму складі кілька типів таймерів, що відрізняються один від одного за функціональним призначенням.

• базові (basictimers)

• загального призначення (general-purposetimers)

• просунуті (advanced-controltimers)

Крім того, в ядро процесора STM32 вбудований 24-бітний системний таймер, так званий SysTick

Перший тип таймерів є найпростішим і являє собою базові таймери (BasicTimers). До даного типу належать таймери TIM6 і TIM7. Ці таймери дуже просто налаштовуються і управляються за допомогою мінімуму регістрів. Вони здатні відраховувати інтервали часу і генерувати переривання при досягненні таймером заданого значення.

Другий тип являє собою таймери загального призначення (General-PurposeTimers). До нього відносяться таймери з TIM2 по TIM5 й таймери з TIM12 по TIM17. Вони можуть генерувати ШІМ, підраховувати імпульси на певних пінах мікроконтролера, обробляти сигнали від енкодера і т.п Третій тип визначає таймери з розвиненим управлінням (Advanced-ControlTimer). До цього типу належить таймер TIM1, який здатний виконувати всі перераховані вище операції. Крім того, на основі даного таймера можна побудувати пристрій, здатний управляти трифазним електропріводом. Таймери з розширеними функціями мають дуже широкий функціонал - комплементарні виводи для підтримки трифазних двигунів, підтримують режими рахунку в прямому і зворотному напрямках, генерацію ШІМ, канали захоплення / порівняння сигналу, режим одиночного імпульсу, підтримка DMA, додаткові функції безпеки у разі збоїв, підтримка інтерфейсу енкодера і датчика Холла.

До шині APB1 підключені таймери TIM2, TIM3, TIM4, TIM5, TIM6, TIM7, TIM12; до шини APB2 - TIM1, TIM8, TIM9, TIM10, TIM11

1. Базові таймери

Базові таймери TIM6 та TIM7 побудовані на основі16-бітових регістрів

Структурна схема базового таймера наведенана рис.5.1 Попередній дільник TIMx_PSC дозволяє регулювати частоту тактових імпульсів і для рахункового регістра, а регіст равтозавантаження TIMx_ARR дає можливість задавати діапазон відліку таймера.

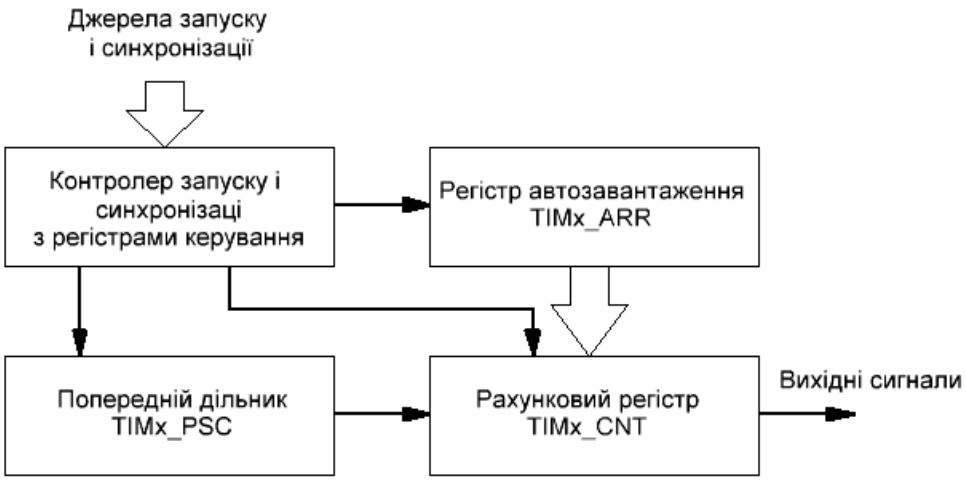


Рис. 5.1 Структурна схема базового таймера

Контролер запуску і синхронізації разом з регістрами управління і стану служить для організації режиму роботи таймера і дозволяє контролювати його функціонування.

Завдяки своїй організації лічильник таймера може рахувати в прямому і в зворотному напрямку, а також до середини заданого діапазону в прямому, а потім у зворотному напрямку. На вхід базового таймера може подаватися сигнал від декількох джерел, у тому числі тактовий сигнал синхронізації від шини APB1, зовнішній сигнал або вихідний сигнал інших таймерів, що подається на висновки захоплення і порівняння.

Таймери TIM6 і TIM7 тактуються від шини APB1. Якщо використовувати кварцовий резонатор з частотою 8 МГц і заводські настройки тактирования за замовчуванням, то тактова частота з шини синхронізації APB1 складе 24 МГц

Регістри TIM6 і TIM7

Карта регістрів наведена на рис.5.2

Карта регистров для базовых таймеров TIM6 и TIM7

Сдвиг	Регистр	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Резерв																								ARPE	Резерв			OPM	URS	UDIS	CEN
	Исх.значение																									0				0	0	0	0
0x04	TIMx_CR2	Резерв																									MMS[2:0]			Резерв				
	Исх.значение																										0	0	0				
0x08	Резерв																																
0x0C	TIMx_DIER	Резерв																							UDE	Резерв							UIE
	Исх.значение																								0								0
0x10	TIMx_SR	Резерв																															UIF
	Исх.значение																																0
0x14	TIMx_EGR	Резерв																															UG
	Исх.значение																																0
0x18	Резерв																																
0x1C	Резерв																																
0x20	Резерв																																
0x24	TIMx_CNT	Резерв																CNT[15:0]															
	Исх.значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	TIMx_PSC	Резерв																PSC[15:0]															
	Исх.значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	TIMx_ARR	Резерв																ARR[15:0]															
	Исх.значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рис.5.2 Карта регістрів

Базові таймери включають наступні 8регістрів:

- TIMx_CNT-Counter(рахунковий регістр);
- TIMx_PSC-Prescaler(попереднійдільник);
- TIMx_ARR-AutoReloadRegister(регістравтоматичного завантаження);
- TIMx_CR1-ControlRegister1(регістр управління 1);

- TIMx_CR2-ControlRegister2(регістр управління 2);
- TIMx_DIER-DMA Interrupt Enable Register (регістр дозволу ПДП іпереривань);
- TIMx_SR-StatusRegister(статусний регістр);
- TIMx_EGR-EventGenerationRegister(реєстргенераціїподій).

РегістриTIMx_CNT, TIMx_PSCiTIMx ARR використовують 16 інформаційних розрядів і дозволяють записувати значення від 0 до 65535.

Частота тактових імпульсів для рахункового регістра TIMx_CNT, що пройшли через дільникTIMx_PSC, розраховується за формулою:

Fcnt = Fin/(PSC + 1),

де Fcnt – частота імпульсіврахунковогорегістра; Fin – тактова частота; PSC – вмістрегіструTIMx_PSC таймеру.

Якщо записати в регістр TIMx_PSC значення 23999, то рахунковий регістр TIMx_CNT при тактовій частоті 24 МГц буде змінювати своє значення 1000 разів в секунду.

Регістр автоматичного завантаження зберігає значення для завантаження рахункового регістра TIMx_CNT. Оновлення вмісту регістра TIMx_CNT проводиться після його переповнення або обнулення, залежно від заданого для нього напрямки рахунку.

Регістр управління TIMx_CR1 має кілька керуючих розрядів

- Розряд ARPE дозволяє і забороняє буферізування запису в регістр автоматичного завантаження TIMx_ARR. Якщо цей біт дорівнює нулю, то при записі нового значення в TIMx_ARR воно буде завантажено в нього відразу. Якщо біт ARPE дорівнює одиниці, то завантаження в регістр відбудеться після події досягнення рахунковим регістром граничного значення

- Розряд OPM включає режим «одного імпульсу». Якщо він встановлений, після переповнення рахункового регістра рахунок зупиняється і відбувається скидання розряду CEN.

- Розряд UDIS дозволяє і забороняє генерування події від таймера. Якщо він обнулений, то подія буде генеруватися при настанні умови генерування події, тобто при переповненні таймера або при програмній установці в регістрі TIMx_EGR розряду UG.

- Розряд CEN включає і відключає таймер. Якщо обнулити цей розряд, то буде зупинений рахунок, а при його установці рахунок буде продовжений. Вхідний дільник при цьому почне рахунок з нуля.

Регістр управління TIMx_CR2 має три керуючих розряду MMS2 ... MMS0, які визначають режим майстра для таймера.

У регістрі TIMx_DIER використовується два розряди.

- Розряд UDE дозволяє і забороняє видавати запит DMA (ПДП) при виникненні події.

- Розряд UIE дозволяє і забороняє переривання від таймера

У регістрі TIMx_SR задіяний тільки один розряд UIF в якості прапора переривання. Він встановлюється апаратно, при виникненні події від таймера. Скидати його потрібно програмно.

Регістр TIMx_EGR містить розряд UG, який дозволяє програмно генерувати подію «переповнення рахункового регістра». При установці цього розряду, відбувається генерація події і скидання рахункового регістра і попереднього дільника. Обнуляється цей розряд апаратно. Завдяки цьому розряду можна програмно генерувати подія від таймера, і тим самим примусово викликати функцію обробника переривання таймера.

2. Приклад використання базового таймеру

Досить часто в процесі програмування виникає завдання реалізації тимчасових затримок. Для вирішення даної задачі необхідна функція формування затримки. Приклад такої функції на основі базової таймера TIM7 для STM32 приведений в лістингу 1.

Лістинг 1

```
#define FAPB1 24000000 // Тактова частота шини APB1
// функція затримки в мілісекундах і мікросекундах
voiddelay(unsignedchar t, unsignedint n)
{
// Завантажити регістр попереднього дільника PSC
If(t==0) TIM7->PSC = FAPB1/1000000-1; // для відліку мікросекунд
If(t==1) TIM7->PSC = FAPB1/1000-1; // для відліку мілісекунд
TIM7->ARR=n; // Завантажити число відліків в регістр автозавантаження ARR
TIM7->EGR |=TIM_EGR_UG; // Згенерувати подію оновлення
// для запису даних в регістри PSC і ARR
TIM7->CR1 |=TIM_CR1_CEN|TIM_CR1_OPM; // Запуск таймеру
// шляхом запису біта дозволу рахунку CEN
// і біту режиму одного проходу OPM в регістр керування CR1
while (TIM7->CR1&TIM_CR1_CEN !=0); // Очікування закінчення рахунку
}
```

Ця функція може формувати затримки в мікросекундах або мілісекундах залежно від параметра «t». Тривалість затримки задається параметром «n». У даній програмі задіяний режим одного проходу таймера TIM7, при якому рахунковий регістр CNT виконує рахунок до значення переповнення, записаного в регістрі ARR. Коли ці значення зрівняються, таймер зупиниться. Факт зупинки таймера очікується в циклі while, шляхом перевірки біта CEN статусного регістра CR1.

Включення тактування таймерів проводиться одноразово в головному модулі програми при їх ініціалізації. Базові таймери підключені до шини APB1, тому подача тактових імпульсів виглядає наступним чином:

```
RCC->APB1ENR |=RCC_APB1ENR_
TIM6EN; // Ввімкнути тактування на TIM6
RCC->APB1ENR |=RCC_APB1ENR_
TIM7EN; // Ввімкнути тактування на TIM7
```

Описаний вище програмний спосіб формування затримки має істотний недолік, пов'язаний з тим, що процесор змушений займатися опитуванням прапора протягом усього часу затримки і тому не має можливості в цей час виконувати інші завдання. Усунути такий недолік можна за допомогою використання режиму переривань від таймера. Функції обробки переривання для базових таймерів зазвичай виглядають наступним чином

```
voidTIM7_IRQHandler ()
{
TIM7->SR&= ~TIM_SR_UIF; // Обнулити прапор
// Виконати операції
}
voidTIM6_DAC_IRQHandler ()
{
// ЯкщоподіявідTIM6
If(TIM6->SR & TIM_SR_UIF)
{
TIM6->SR &= ~TIM_SR_UIF; //Обнулитипрапор
```

```
// Виконатиоперації
}
}
```

3. Таймери загального призначення (General-PurposeTimers).

Таймери **General-Purpose Timers** мікроконтролера STM32F4xx мають більш широкі можливості, ніж базові, наприклад мають режими ШІМ, захоплення події, порівняння, тощо.

До таймерів **General-Purpose Timers** відносяться таймери з TIM2 по TIM5 і таймери з TIM12 по TIM17. Тактування таймерів TIM2-TIM7, TIM12-TIM14 здійснюється генератором APB1; таймерів TIM1, TIM8-11 – APB2.

Тема 3.6 Контролери NVIC та EXTI

1. Джерела переривань NVIC та EXTI

Управління та обробка перериваннями виробляється контролером пріоритетних векторних переривань NVIC (Nested Vectored Interrupt Controller). Джерела запитів переривань контролера наведено на рис.6.1

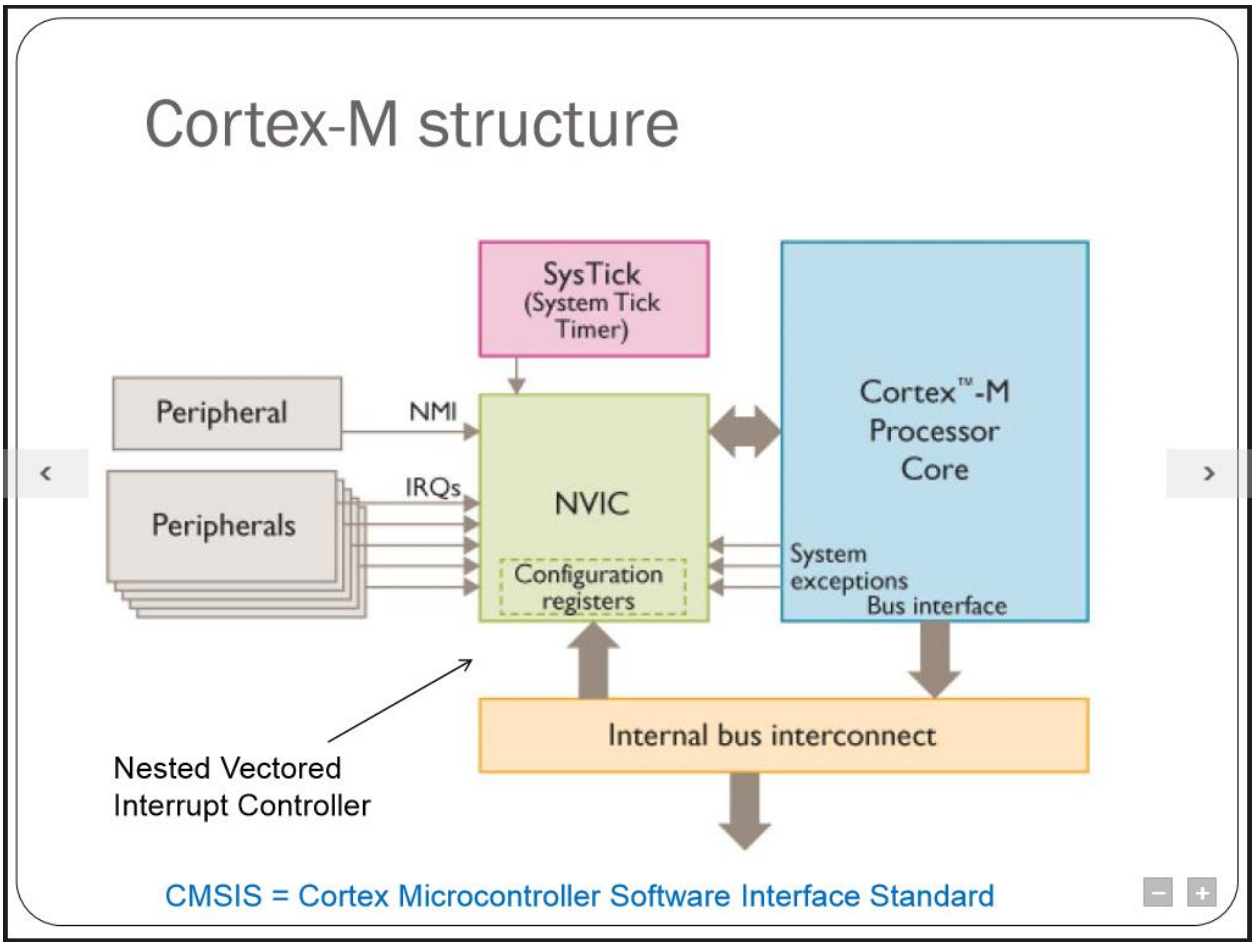


Рис.6.1 Джерела запитів переривань контролера

Як видно, на контролер NVIC приходять переривання від периферійних пристроїв МК, ядра і таймера.

Зовнішні переривання, які поступають на піни МК, обробляються за допомогою використовується контролера зовнішніх переривань/ подій EXTI (External interrupt / event controller) Взаємодія його з NVIC ілюструє рис.6.2

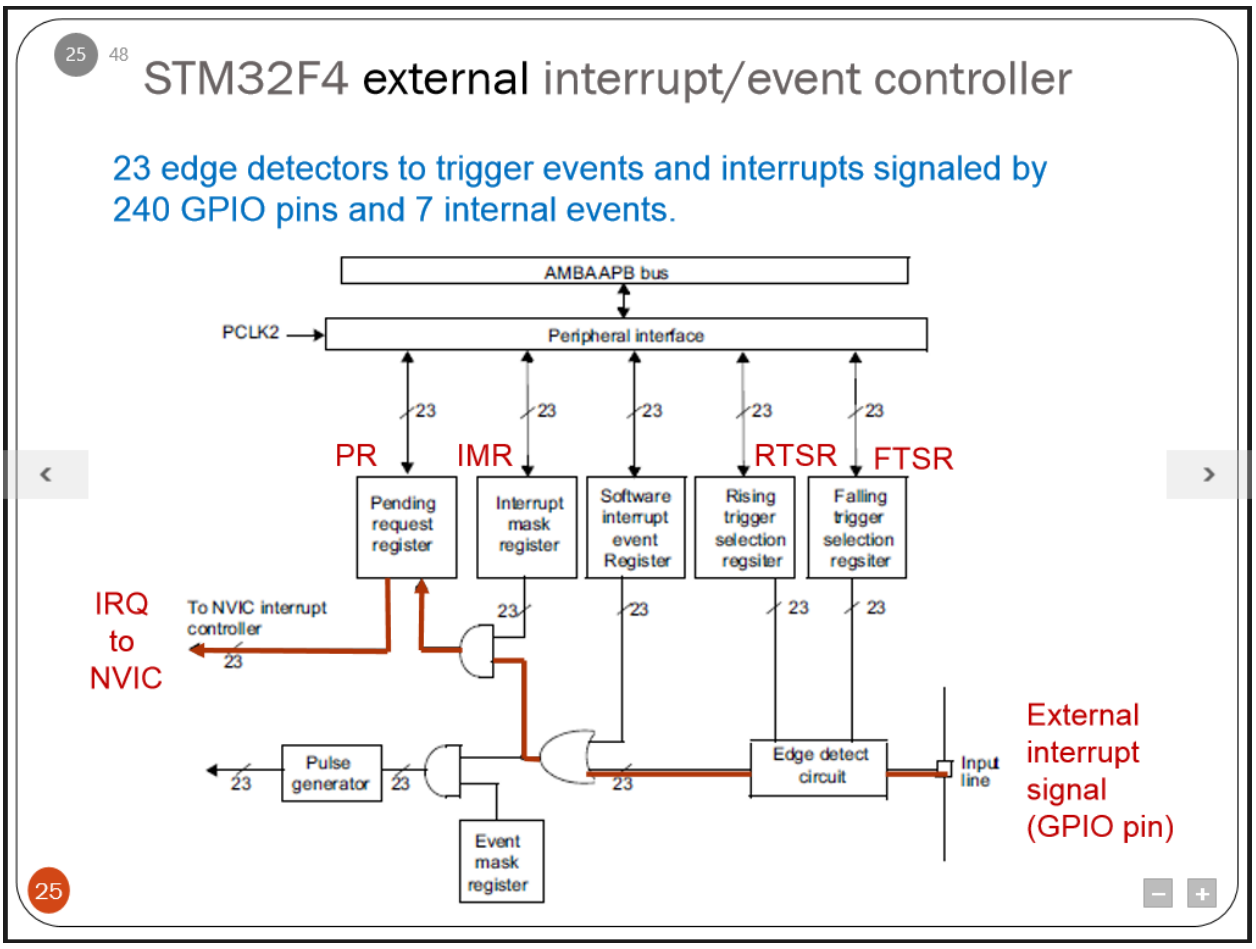


Рис.6.2 Взаємодія мікроконтролера з NVIC

Зовнішнє переривання з GPIOpin через блок вибору детектування переднього або заднього фронту, схеми маскування та обробки режиму очікування поступає на вхід контролера пріоритетних векторних переривань NVIC.

2. Переривання та події (Interrupt and Events)

В МК Cortex-M є два поняття, які часто плутають - Interrupt і Event(переривання та подія).

- **Event** - це подія (апаратна або програмна), на яку можуть реагувати ядро або периферійні блоки. Одним з варіантів реакції може бути - переривання.
- **Interrupt** - це переривання роботи програми і перехід управління на спеціалізований ділянку - обробник переривання.

Кожне переривання викликається подією, але не кожна подія викликає переривання. Крім переривань, події можуть активувати і інші можливості МК, рис.6.3

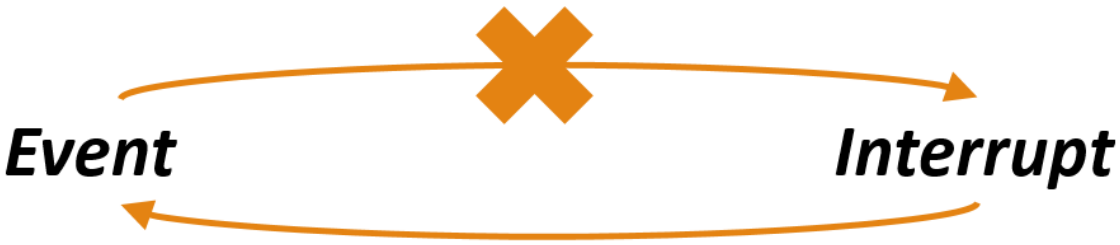


Рис. 6.3 Інші можливості МК

Характеристика контролера NVIC

- Загальна кількість переривань 240 (0-239)
- Програмований рівень пріоритету 0-255 для кожного переривання. Більш високий рівень відповідає нижчому пріоритету, тому рівень 0 має найвищий пріоритет переривання.
- Сигнали переривань можуть задаватися як рівнем, так і фронтами імпульсів.
- Динамічна зміна пріоритетів переривань.
- Угрупування пріоритетів по групам та підгрупам.
- Переривання у «хвості ланцюжка».

- Зовнішнє немасковане переривання NonMaskableInterrupt (NMI)

Додатковий контролер WIC (WakeupInterruptController) забезпечує підтримку режиму сну з ультра-низьким енергоспоживанням..

Процесор автоматично зберігає свій стан на вході переривань і витягує його зі стеку на виході без додаткових команд Це забезпечує латентність обробки виключень

3. Стани переривань

Кожен перериванняможе знаходиться в одному з наступних станів:

- Inactive (Неактивне): не активно і не прийнято.
- Pending (В очікуванні): чекає обслуговування процесором.
- Active (Активне): обслуговується процесором, але обслуговування не завершено
- Active and pending (Активне і в очікуванні) обслуговується процесором, але в той же часочікується виключення від того ж джерела.

4. Вхід в перериванняі вихід з нього

При виникненні деякої події контролер переривань автоматично перериває виконання основної програми, і викликає відповідну функцію обробки переривань. Після виходу з функції обробника переривань програма продовжує виконання з того місця, де відбулося переривання, рис. 6.4

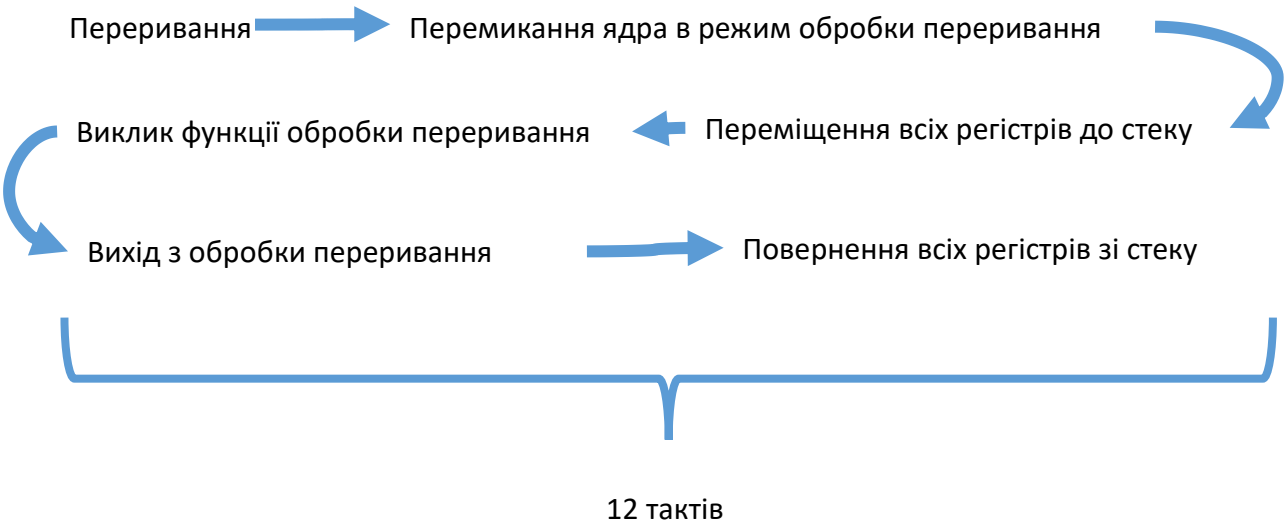


Рис.6.4

Після перемикання ядра в режим обробки перериваннярегістри ядра поміщаються в стек. Безпосередньо під час запису значення регістрів в стек здійснюється вибірка початкової адреси функції обробки переривання.

В стек переміщається наступні регістри - регістр статусу програми (Program Status Register (PSR)), лічильник програми (Program Counter (PC)) і регістр зв'язку (Link Register (LR)).(Опис регістрів ядра приведено в Cortex-M4 Generic User Guide та 2 лекції)Таким чином запам'ятовується стан, в якому перебувало ядро перед переходом в режим обробки переривань.

Також зберігаються регістри R0 - R3 і R12. Ці регістри використовуються в інструкціях для передачі параметрів, тому переміщення їх в стек робить можливим їх використання у функції обробки переривання, а R12 часто виступає в ролі робочого регістра програми.

По завершенні обробки переривання всі дії виконаються в зворотному порядку: витягується вміст стека і, паралельно з цим, здійснюється вибірка адреси повернення.

З моменту ініціації переривання до виконання першої команди обробника перериваних проходить 12 тактів, такий же час необхідно для

поновлення основної програми після завершення обробки переривання (див. рис.6.4).

5. Вкладеність переривань

Контролер NVIC підтримує вкладеність переривань і пріоритети. Кожному перериванню при налаштуванні NVIC присвоюється свій пріоритет. Якщо під час обробки низько пріоритетного переривання виникає високо пріоритетне, то воно, в свою чергу, зупинить обробник низькопріоритетного переривання.

NVIC підтримує переривання з різними пріоритетами, які можуть переривати один одного. При цьому, можуть виникнути різні ситуації, обробка яких по різному оптимізована:

1. Призупинення низькопріоритетного переривання

У цій ситуації, обробка низькопріоритетного переривання припиняється. Наступні 12 циклів виконується збереження в стек нового набору даних і запускається обробка високопріоритетного переривання. Після його обробки, вміст стека автоматично витягується і поновлюється обробка низькопріоритетного переривання.

2. Безперервна обробка переривань

Ця ситуація може виникнути в двох випадках: якщо два переривання мають однаковий пріоритет і виникають одночасно або якщо низькопріоритетне переривання виникає під час обробки високопріоритетного.

В цьому випадку, проміжні операції над стеком не здійснюються. Відбувається тільки завантаження адреси обробника наступного переривання і перехід до його виконання. Відмова від операцій над стеком економить 6 тактів, тобто перехід відбувається не за 12 тактів, а за 6.

3. Запізнення високопріоритетного переривання

Ситуація виникає, якщо високопріоритетне переривання відбувається під час переходу до низькопріоритетного. В цьому випадку перехід до високопріоритетного переривання відбуватиметься не менш ніж за 6 тактів з моменту його виникнення (час необхідний для завантаження адреси обробника переривання і переходу до нього).

Випадки 2 та 3 - переривання у «хвості ланцюжка»- пояснюється рис. 6.5

Два переривання IRQ1 IRQ2 прийшли одночасно, IRQ1 внаслідок більш високого пріоритету отримала обслуговування. При поверненні із підпрограми IRQ1 і переході до підпрограми обробки IRQ2 стан основної програми не має сенсу зберігати в стеку і перехід відбувається безпосередньо.

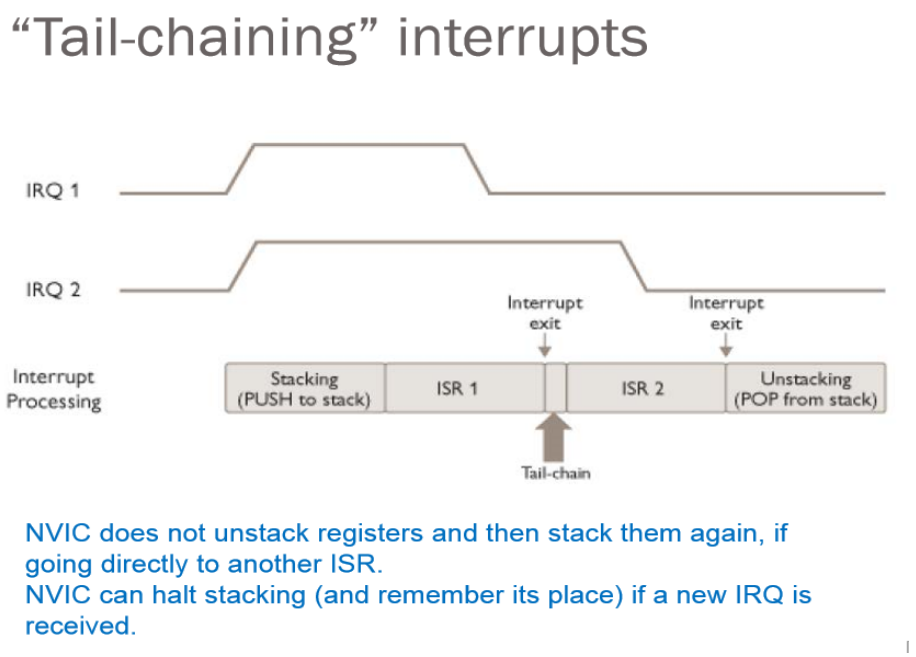


Рис.6.5

6. Пріоритети переривань

Крім простої установки пріоритету переривань, NVIC реалізує можливість угрупування пріоритетів.

Переривання в групі з більш високим пріоритетом можуть переривати обробники переривань групи з нижчим пріоритетом. переривання з однієї групи, але з різним пріоритетом всередині групи не можуть переривати один одного. Пріоритет всередині групи визначає тільки порядок виклику обробника, коли були активізовані обидві події.

7. Маскування переривань

Для того, щоб вмикати / вимикати різні вектора переривань, існує маскування переривань.

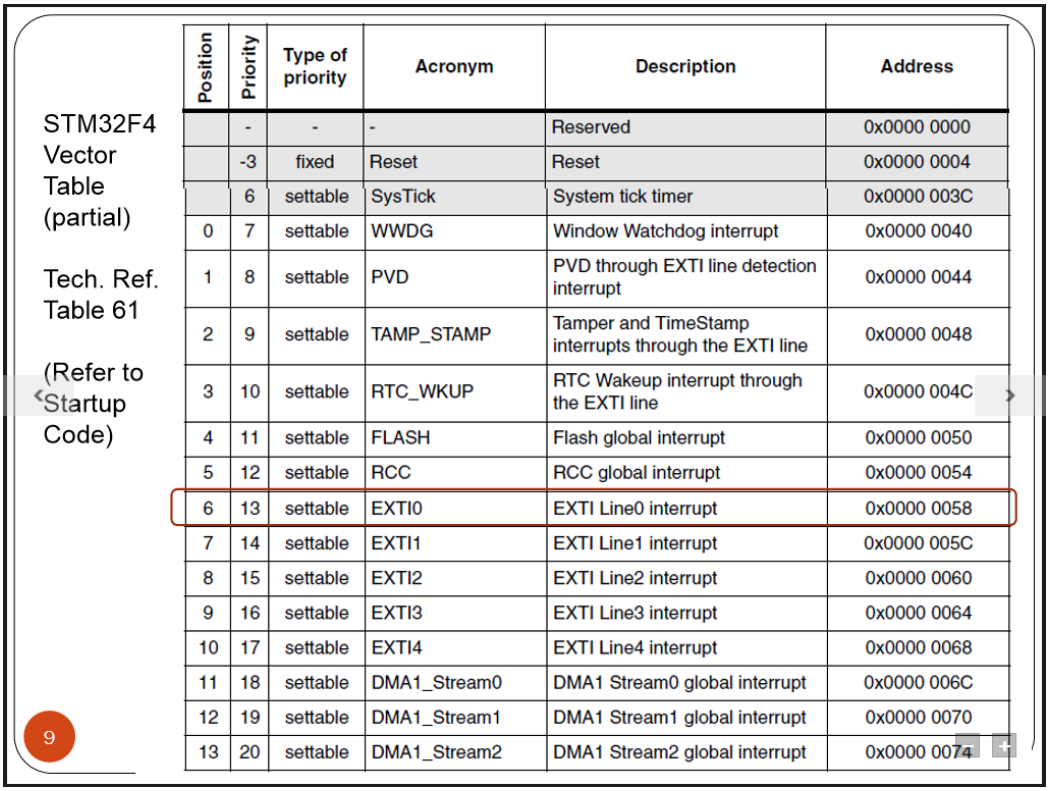
Маскування переривання здійснюється за допомогою регістрів Interrupt Set-enable Registers.

Якщо переривання замасковано, це не означає, що периферія не генерує події! Просто NVIC не викликає обробник цієї події.

8. Таблиця векторів переривань

Таблиця векторів переривань містить адреси функцій обробників переривань і знаходиться в області пам'ят за 0 адресою. Можна розмістити таблицю векторів переривань в іншій області пам'яті За це зміщення таблиці векторів відповідає регістр таблиці векторів зсуву. Номер у списку відповідає номеру переривання. Таблицю векторів можна знайти для конкретного контролера в STM32F10x.s файлі:

Таблиця 6.1



STM32F4 Vector Table (partial)

Tech. Ref. Table 61

(Refer to Startup Code)

Position	Priority	Type of priority	Acronym	Description	Address
	-	-	-	Reserved	0x0000 0000
	-3	fixed	Reset	Reset	0x0000 0004
	6	settable	SysTick	System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	PVD	PVD through EXTI line detection interrupt	0x0000 0044
2	9	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	0x0000 004C
4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068
11	18	settable	DMA1_Stream0	DMA1 Stream0 global interrupt	0x0000 006C
12	19	settable	DMA1_Stream1	DMA1 Stream1 global interrupt	0x0000 0070
13	20	settable	DMA1_Stream2	DMA1 Stream2 global interrupt	0x0000 0074

9

Можна розмістити таблицю векторів переривань в іншій області пам'яті За це зміщення таблиці векторів відповідає регістр таблиці векторів зсувуVector Table Offset Register (VTOR)

9. Регістри NVIC

Регістри NVIC наступні

ISER - Interrupt Set Enable Register. NVIC_ISER0-NVIC_ISER7

Регістри дозволу переривань

ICER - Interrupt Clear-enable RegistersNVIC_ICER0-NVIC_ICER7

Регістри заборони переривань

ISPR - Interrupt Set Pending Register. NVIC_ISPR0-NVIC_ISPR7. Регістри постановки в очікування

ICPR - Interrupt Clear-pending Registers. NVIC_ICPR0-NCVIC_ICPR7 Зняти переривання з очікування.

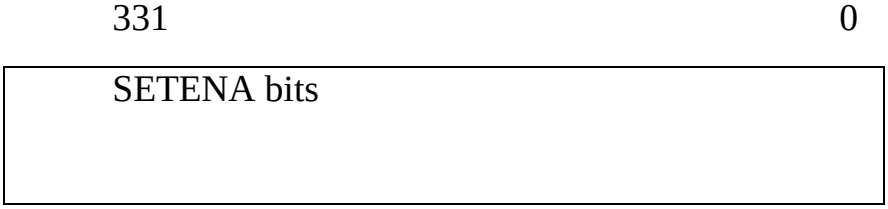
1. **IABR** - Interrupt active bit registers. NVIC_IABR0-NVIC_IABR7

Вказівники активних переривань

2. IPR - Interrupt Priority Registers registers.NVIC_IPR0-NVIC_IPR59 Регістри пріоритетів

3. **STIR -** Software Trigger Interrupt RegisterРегістр викликів програмного забезпечення переривань

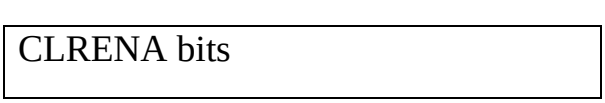
Регістри дозволу переривань ISER- Interrupt Set Enable Register (NVIC_ISER0-NVIC_ISER7)



Таблиця 6.2

Біти	**Назва**	**Функція**
[31:0]	SETENA	Біти дозволу переривань. запис: 0 = не робить нічого 1 = дозвіл переривання. читання: 0 = переривання заборонено 1 = переривання дозволено

Регістри заборони переривань ICER - Interrupt Clear-enable Registers. (NVIC_ICER0-NVIC_ICER7)



Таблиця 6.3

Біти	Назва	Функція
[31:0]	CLRENA	Біти заборони/дозволу переривань. запис: 0 = не робить нічого 1 = заборона переривання. читання: 0 = переривання заборонено 1 = переривання дозволено

Регістри постановки в очікування **ISPR** - Interrupt Set Pending Register. NVIC_ISPR0-NVIC_ISPR7. Переводять переривання в відкладений стан і показують, яке переривання знаходиться в очікуванні.

31	0
SETPEND bits	

Таблиця 6.4

Біти	Назва	Функція
[31:0]	SETPEND	Біти постановки в очікування. запис: 0 = не робить нічого 1 = змінює стан передання – ставить в очікування. читання: 0 = переривання не в стані очікування

Біти	Назва	Функція
		1 = переривання в стані очікування

Примітка

Запис 1 в біт ISPR, відповідно:

• для переривання, що в стані очікування, не має ніякого ефекту

• для забороненого переривання, встановлює стан цього переривання в стан очікування.

Регістри зняття переривання з очікуванняICPR - Interrupt Clr Pending Register (NVIC_ICPR0-NCVIC_ICPR7)

31 0

CLRPEND bits

Таблиця 6.6

Біти	Назва	Функція
[31:0]	CLRPEND	Біти зняття переривання з очікування. запис: 0 = не робить нічого 1 = знімає переривання з очікування. читання: 0 = переривання не в стані очікування 1 = переривання в стані очікування

Примітка

• Запис 1 до ICPR не впливає на активний стан відповідного переривання.

Вказівники активних переривань IABR - Interrupt active bit registers (NVIC_IABR0-NVIC_IABR7). Регістр показує чи активно в даний момент переривання. Автоматично встановлюється на початку обробки переривання і автоматично же знімається під час виходу з нього. Цей регістр можна тільки читати.

31	0
ACTIVE bits	

Таблиця 6.7

Біти	**Назва**	**Функція**
[31:0]	ACTIVE	Прапори активних переривань. 0 = переривання не активне 1 = переривання активне

Біт прочитується як одиничний , якщо статус відповідного переривання активний або переривання знаходиться у стані очікування .

Регістри пріоритетів IPR - Interrupt Priority Registers registers (NVIC_IPR0-NVIC_IPR59.Задає значення пріоритету переривання (0-255)

	31 24	23 16	15 8	7 0
IPR59	PRI_239	PRI_238	PRI_237	PRI_236
…				
IPR*n*	PRI_4n+3	PRI_4n+2	PRI_4n+1	PRI_4n
…				
IPR0	PRI_3	PRI_2	PRI_1	PRI_0

Таблиця 6.8

Біти	Назва	Функція
[31:24]	Priority, byte offset 3	Кожне пріоритетне поле може мати значення 0-255. Чим нижче значення, тим більше пріоритет відповідного переривання Ширина кожного поля 8 біт.Невизначені біти читаються як 0 і ігноруються при запису
[23:16]	Priority, byte offset 2	
[15:8]	Priority, byte offset 1	
[7:0]	Priority, byte offset 0	

Зауважимо, що хоча теоретично пріоритетів може бути 256, на практиці реально використовують тільки старші 4 біта, утворюючи таким чином 16 рівн

Регістр викликів програмного забезпечення переривань STIR (SoftwareTriggerInterruptRegister)

Запис в STIR генерує переривання від програмного забезпечення.

31 9	8 0
Reserved	INTID

Таблиця 6.9

Біти	Поля	Функція
[31:9]	-	Зарезервовано
[8:0]	INTID	Ідентифікатор переривання, щоб викликається, лежить в діапазоні 0-239.

Біти	Поля	Функція
		Наприклад, значення 0x03 вказує на переривання IRQ3.

9. Використання бібліотечних функцій

Програмне забезпечення використовує інструкції CPSIE I і I CPSID для включення і відключення переривань.

CMSIS надає наступні вбудовані функції для цих інструкцій:

void __disable_irq(void) // Disable Interrupts

void __enable_irq(void) // Enable Interrupts

Крім того, CMSIS надає ряд функцій для контролю NVIC, у тому числі:

Опис CMSIS функцій доступу до NVIC регістрів

Таблиця 6.10

CMSIS функція	**Опис**
void NVIC_EnableIRQ(IRQn_Type IRQn)	Включає переривання
void NVIC_DisableIRQ(IRQn_Type IRQn)	Відключає переривання
void NVIC_SetPendingIRQ(IRQn_Type IRQn)	Встановлює статус очікування переривання: 1.
void NVIC_ClearPendingIRQ(IRQn_Type IRQn)	Очищає відкладений статус переривання: 0.
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)	Читає відкладений статус переривання. Ця функція повертає ненульове

CMSIS функція	**Опис**
	значення, якщо стан очікування встановлений в 1.
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)	Встановлює пріоритет переривання з налаштованим рівнем пріоритету: 1.
uint32_t NVIC_GetPriority(IRQn_Type IRQn)	Читає пріоритет переривання або з налаштованим рівнем пріоритету. Ця функція повертає поточний рівень пріоритету.

Контролер зовнішніх переривань EXTI (External interrupt/event controller)

Апаратні переривання

Контролер зовнішніх подій \ переривань обробляє 23 лінії від датчиків, що генерують переривання:

EXTI0…15 конфігурується к портам GPIO

EXTI16 -> PVD output

EXTI17 -> RTC Alarm event

EXTI18 -> USB OTG FS Wakeup event

EXTI19 -> Ethernet Wakeup event

EXTI20 -> USB OTG HS (configured in FS) Wakeup event

EXTI21 -> RTC Tamper and TimeStamp events

EXTI22 -> RTC Wakeup event

Регістри EXTI

1. Регістр маски Interrupt mask register (EXTI_IMR)
2. Регістр вибору наростаючого фронту Rising trigger selection register (EXTI_RTSR)
3. Регістр вибору спадаючого фронту Falling trigger selection register (EXTI_ FTSR)
4. Регістр очікування Interrupt/event pending register (EXTI_PR)
5. Регістр програмного виклику зовнішнього переривання Software interrupt event register (EXTI_SWIER)
6. Регістр керування/стану перериванньInterrupt Control and State Register (ICSR)
7. Регістр управління угрупованням пріорітетів Application Interrupt and Reset Control Register (AIRCR)

Регістр маски Interrupt mask register (EXTI_IMR)

Таблиця 6.11

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Біти 31:23 Зарезервовані

Біти 22:0 MRx: маскування лінії x

0: Переривання по лінії x замасковано

1: Переривання по лінії x не замасковано

Регістр вибору наростаючого фронту Rising trigger selection register (EXTI_RTSR)

Таблиця 6.12

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **TRx:** Rising trigger event configuration bit of line x
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

Регістр вибору спадаючого фронту Falling trigger selection register (EXTI_ FTSR)

Формат аналогічний EXTI_RTSR, однак 1 у відповідному біті обирає спадаючий фронт імпульсу для виклику зовнішнього переривання на лінії. 0 – ігнорує тип фронту

Регістр очікування Interrupt/eventpendingregister (EXTI_PR) – відповідний біт читається як 1, якщо переривання/подія відбулася; запис 1 скидає стан очікування (запис 0 не має ніякого ефекту);

Регістр програмного виклику зовнішнього переривання Software interrupt event register (EXTI_SWIER) - 1 встановлює біт очікування в регістрі PR :Виклик переривання не маскується

Регістр керування/стану перериванъ Interrupt Control and State Register (ICSR)

• забезпечує:

- Встановлення очікування бітам за винятком NMI переривання
- Встановлення/скидання бітів очікування для PendSV і SysTick виключень

• вказує:

- номер виключення, що обробляється
- майбутні активні виключення
- номер винятку з найвищим пріоритетом відкладеного винятку абопро будь- яке переривання в очікуванні

Призначення бітів регістру **ICSR**

Таблиця 6.13

Біти	**Назва**	**Тип**	**Функція**
[31]	NMIPENDSET	RW	NMI set-pending bit. Write: 0 = no effect 1 = changes NMI exception state to pending. Read: 0 = NMI exception is not pending 1 = NMI exception is pending. Because NMI is the highest-priority exception, normally the processor enter the NMI exception handler as soon as it registers a write of 1 to this bit, and entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.
[30:29]	-	-	Reserved.
[28]	PENDSVSET	RW	PendSV set-pending bit. Write:

Біти	Назва	Тип	Функція
			0 = no effect 1 = changes PendSV exception state to pending. Read: 0 = PendSV exception is not pending 1 = PendSV exception is pending. Writing 1 to this bit is the only way to set the PendSV exception state to pending.
[27]	PENDSVCLR	WO	PendSV clear-pending bit. Write: 0 = no effect 1 = removes the pending state from the PendSV exception.
[26]	PENDSTSET	RW	SysTick exception set-pending bit. Write: 0 = no effect 1 = changes SysTick exception state to pending. Read: 0 = SysTick exception is not pending 1 = SysTick exception is pending.
[25]	PENDSTCLR	WO	SysTick exception clear-pending bit.

Біти	Назва	Тип	Функція
			Write: 0 = no effect 1 = removes the pending state from the SysTick exception. This bit is WO. On a register read its value is `Unknown`.
[24]	-	-	Reserved.
[23]	Reserved for Debug use	RO	This bit is reserved for Debug use and reads-as-zero when the processor is not in Debug.
[22]	ISRPENDING	RO	Interrupt pending flag, excluding NMI and Faults: 0 = interrupt not pending 1 = interrupt pending.
[21:18]	-	-	Reserved.
[17:12]	VECTPENDING	RO	Indicates the exception number of the highest priority pending enabled exception: 0 = no pending exceptions Nonzero = the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.

Біти	Назва	Тип	Функція
[11]	RETTOBASE	RO	Indicates whether there are preempted active exceptions: 0 = there are preempted active exceptions to execute 1 = there are no active exceptions, or the currently-executing exception is the only active exception.
[10:9]	-	-	Reserved.
[8:0]	VECTACTIVE[a]	RO	Contains the active exception number: 0 = Thread mode Nonzero = The exception number[a] of the currently active exception. Subtract 16 from this value to obtain the CMSIS IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers.

Регістр управління угрупованням пріорітетів Application Interrupt and Reset Control Register (AIRCR)

Регістр забезпечує управління угрупованням, представленням даних (MSB- LSB) і скиданням контролю над системою.

Таблиця 6.14

3116	15	14 11	10 8	7 3	2	1	0

On read: VECTKEYSTATOn write: VECTKEY	Reserved		Reserved				

Біти відображені в табл.6.14:

Таблиця 6.15. ***AIRCR bit assignments***

Біти	**Назва**	**Тип**	**Функція**
[31:16]	Write: VECTKEYSTAT Read: VECTKEY	RW	Register key: Reads as `0xFA05` On writes, write `0x5FA` to VECTKEY, otherwise the write is ignored.
[15]	ENDIANNESS	RO	Data endianness bit is implementation defined: 0 = Little-endian 1 = Big-endian.
[14:11]	-	-	Reserved.
[10:8]	PRIGROUP	R/W	Interrupt priority grouping field is implementation defined. This field determines the split of group priority from subpriority, see*Binary point*.
[7:3]	-	-	Reserved.
[2]	SYSRESETREQ	WO	System reset request bit is implementation defined: 0 = no system reset request 1 = asserts a signal to the outer system that requests a reset. This is intended to force a large system reset of all major components except for debug.

Біти	Назва	Тип	Функція
			This bit reads as 0. See you vendor documentation for more information about the use of this signal in your implementation.
[1]	VECTCLRACTIVE	WO	Reserved for Debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.
[0]	VECTRESET	WO	Reserved for Debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.

Поле **PRIGROUP** вказує положення двійковій точки, який розділяє поля PRI_n в регістрах InterruptPriorityRegisters на окремі пріоритетні групи і підгрупи. Таблиця 6.15 показує, як елементи управління PRIGROUP це реалізують

*Таблиця 6.15**. **Priority grouping***

	Interrupt priority level value, PRI_*N*[7:0]			**Number of**	
PRIGROUP	**Binary point**	**Group priority bits**	**Subpriority bits**	**Group priorities**	**Subpriorities**
0b000	bxxxxxxx.y	[7:1]	[0]	128	2
0b001	bxxxxxx.yy	[7:2]	[1:0]	64	4
0b010	bxxxxx.yyy	[7:3]	[2:0]	32	8
0b011	bxxxx.yyyy	[7:4]	[3:0]	16	16
0b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
0b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
0b110	bx.yyyyyyy	[7]	[6:0]	2	128
0b111	b.yyyyyyyy	None	[7:0]	1	256

Пріоритет групи визначає, чи може одне переривання перебивати інше. Скажімо, переривання групи 0 легко забирають керування у переривань групи 1.

А в кожній групі виникають підгрупи пріоритетів. Вони уже вирішують, хто піде першим, якщо одночасно прийдуть на виконання два переривання однієї групи. Але перебивати один одного переривання з однієї групи вже не можуть. А якщо одночасно прийдуть два переривання з однієї групи і однієї підгрупи, то виклик піде по порядку їх описання в таблиці векторів.

Група визначається в спеціальному поля PRIGROUP регістраAIRCR, записавши туди число від 0 до 7 (для STM32F103 лише від 3 до 7). При цьому поле задання пріоритету в IRPділиться на групу і підгрупу.

11. Налагодження переривань EXTI

Для настройки 23 ліній як джерел переривань, використовують наступну процедуру:

• Налаштування маски бітів для 23 ліній переривань (EXTI_IMR)

• Налаштування детектор фронтів за допомогою регістрів EXTI->RTSR і EXTI->FTSR

• Налаштування дозволу/заборони маски бітів, які контролюють канал NVICIRQ, відображений на зовнішній контролер переривань (EXTI).

Приклад

Нехай необхідно налаштувати переривання, які будуть спрацьовувати при переході контакту PA0 зі стану «0» в стан «1», і переривання при переході контакту PC6 зі стану «1» до стану «0».

Будемо використовувати 0 і 6 лінії EXTI. Для розмаскування відповідних ліній переривань необхідно записати в регістр EXTI_IMR значення 0x9.

Для лінії PA0, необхідна генерація події переривання по переходу зі стану «0» в стан «1» - по зростаючому фронту. Тобто, необхідно записати 1 в нульовий біт регістра EXTI_RTSR.

Для лінії PC6, навпаки, необхідна генерація події переривання по переходу зі стану «1» до стану «0» - по падаючому фронту. Тобто, необхідно записати 1 в шостий біт регістра EXTI_FTSR.

На цьому налаштування блоку EXTI закінчена. Останній пункт буде реалізований при налагодженні NVIC.

11. Нлагодження NVIC

Активація обробки певного вектора переривання здійснюється за допомогою регістрів**ISER**Interruptset-enableregisters (NVIC_ISERx). Таблиця векторів переривань наведена в Referencemanual (див. Таблиця 6.16).

Таблиця 6.16

Position	Priority	Type of priority	Acronym	Description	Address
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C

З таблиці видно, що для 0 лінії є окреме переривання (6 біт (позіція) **ISER)**, а лінії з 5 по 9 генерують одне переривання на всіх (23 біт)

Крім того, з таблиці знаходимо номери векторів, необхідних переривань. Тепер потрібно записати «1» в 6 біт (активація переривань лінії 0 EXTI) регістра NVIC_ISER0 (адреса 0xE000E100) і в того ж регістра (активація переривань ліній 5-9).

12. Налагодження пріоритетів

Для дослідження впливу пріоритетів того, чтобы можно было побаловаться с приоритетами прерываний налаштуємо групи пріоритетів так, щоб 2 біта відповідали за пріоритет всередині групи, і 2 біта - за пріоритет самої групи. Для цього необхідно записати значення 0x05FA0500 в регістр

Application interrupt and reset control register (Налаштування пріоритетів здійснюється за допомогою регістрів **Interrupt Priority Registers.** В даному прикладі будуть використані регістри InterruptPriorityRegister 2 (0xE000E4008) і регістр InterruptPriorityRegister 5 (0xE000E401C).

Встановимо пріоритетиоднакові для обох переривань.

13. Обробка переривань

Після закінчення обробки переривання необхідно скинути активність події, що викликало переривання - «очистити переривання». Очищення переривання EXTI здійснюється за допомогою регістра EXTI_PR. Зверніть увагу: запис «1» очищає переривання, запис «0» не має жодного впливу.

Якщо з обробкою переривання лінії 0 EXTI все досить просто, то з групою ліній 5-9 виникає питання - як визначити яка лінія викликала переривання. Дізнатисяцеможнаперевіркоюбітрегістра Pending register (EXTI_PR) - Reference manual.

Створюємо таблицю векторів і розташовуємо її в правильному місці

Для того, щоб таблиця векторів з правильними адресами функцій обробників переривань розташовувалася на початку флеш пам'яті МК, створимо і підключимо до проекту файл startup.c.

Він має у своєму розпорядженні таблицю __vector_table на початку секції, оголошеної в файлі лінкера. Сам файл можна подивитися тут:

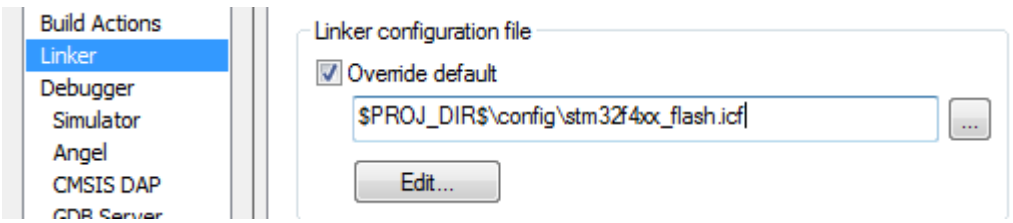


Рис.6.6

Сама секція задається на початку ROM пам'яті. Адреси можна подивитися тут (документ, в якому описано адресація флеш пам'яті STM32):

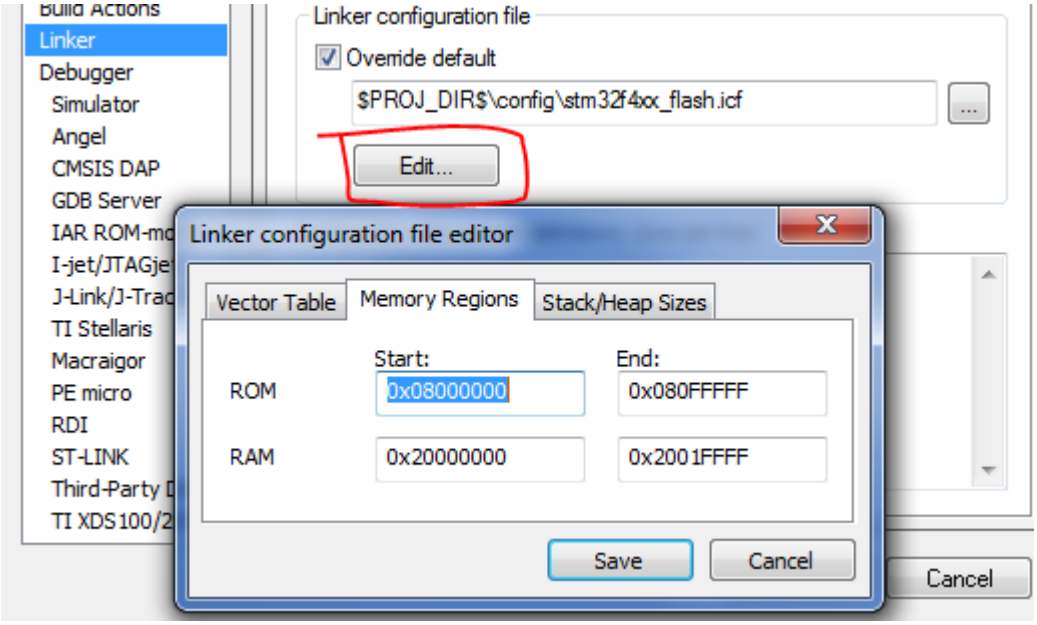


Рис.*6.7*

Комбінація директиви IAR і спец функції IAR:
#pragma segment="CSTACK"

__sfe("CSTACK")

Записує на початку флеша покажчик на верхівку стека.

Саму таблицю заповнюють адреси функцій, які реалізують вічний цикл. Виняток зроблено лише для функцій, що нас цікавлять:

externvoid EXTI_Line0_IntHandler(void);

externvoid EXTI_Line6_IntHandler(void);

У функції, що викликається при старті, просто виконується перехід до externvoid __iar_program_start(void);

Це функція - main (). Сам символ можна перевизначити, якщо виникне бажання:

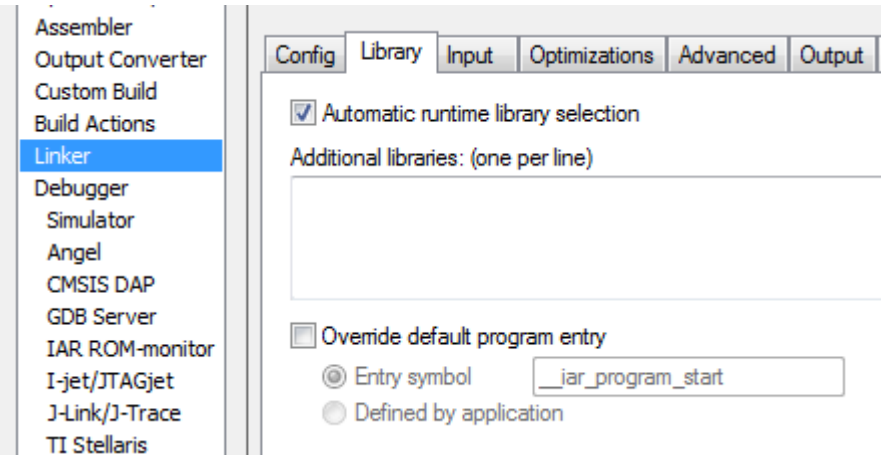


Рис.6.8

Переходимо до основного файлу

Для початку випишемо і перевизначимо всі адреси і бітові поля, які нам знадобляться.

Зверніть увагу на те, що значення спецрегістрів МК оголошені як volatile. Це необхідно, щоб компілятор не намагався оптимізувати операції звернення до них, оскільки це не просто ділянки пам'яті і їх значення можуть змінюватися без участі ядра.

Налаштовуємо групування пріоритетів

В першу чергу варто налаштувати угруповання пріоритетів переривань:

```
SCB_AIRCR = SCB_AIRCR_GROUP22;
```

Дана дія повинна виконуватися тільки один раз. У складних проектах, що використовують сторонні бібліотеки варто перевіряти даний факт. Зміна розбиття пріоритетів на групи може привести до некоректної роботи прошивки.

Включення тактування використовуваної периферії

Нагадаю, що перед початком роботи з периферійними блоками необхідно включити їх тактування:

```
//Enable SYSCFG , GPIO port A and D clocking
RCC_AHB1_ENR |= RCC_AHB1_ENR_GPIOA|RCC_AHB1_ENR_GPIOC|RCC_AHB1_ENR_GPIOD;
RCC_APB2_ENR |= RCC_APB2_ENR_SYSCFG;
```

Працювати відразу з SYSCFG не можна, потрібно почекати кілька тактів. Але ми і не будемо. Займемося инициализацією GPIO.

Ініціалізація GPIO

Світлодіоди не ініціалізуються так само як і минулого разу.

```
//LED3 and LED5 initialization
GPIOD_MODER = (GPIOD_MODER & (~GPIO_MODER_13BITS)) | GPIO_MODER_13OUT;
GPIOD_MODER = (GPIOD_MODER & (~GPIO_MODER_14BITS)) | GPIO_MODER_14OUT;
```

Кнопка PA0 і контакт PC7 ініціалізуються як вхідні:

```
//PA0 and PC6 pins initialization
GPIOA_MODER = (GPIOA_MODER & (~GPIO_MODER_0BITS)) | GPIO_MODER_0IN;
GPIOC_MODER = (GPIOC_MODER & (~GPIO_MODER_6BITS)) | GPIO_MODER_6IN;
```

Ось тільки для контакту PC6 необхідно включити підтяжку живлення. Активація підтяжки проводиться за допомогою регістра GPIOC_PUPDR:

```
//Enable PC6 pull-up
GPIOC_PUPDR = (GPIOC_PUPDR & (~GPIOC_PUPDR_7BITS)) | GPIO
C_PUPDR_6PU;
```

14. Налаштування EXTI

Отже, нам потрібно налаштувати наступні параметри - включити переривання для ліній 0 і 6, для лінії 0 переривання по зростаючому фронту, для лінії 6 - переривання по падаючому фронту:

```
//Set up EXTI
EXTI_RTSR |= EXTI_LINE0;
EXTI_FTSR |= EXTI_LINE6;
EXTI_IMR  = EXTI_LINE0|EXTI_LINE6;
```

Залишилося налаштувати піни яких портів підключені до лінії EXTI (дивне рішення, наприклад МК stellaris можуть генерувати переривання при будь-якій комбінації пінів, у STM32 з цим складніше):

```
//EXTI to port  connection
SYSCFG_EXTICR1  =  (SYSCFG_EXTICR1&(~SYSCFG_EXTICR1_0BI
TS)) | SYSCFG_EXTICR1_0PA;
SYSCFG_EXTICR2  =  (SYSCFG_EXTICR2&(~SYSCFG_EXTICR2_6BI
TS)) | SYSCFG_EXTICR2_6PC;
```

15. Налаштування NVIC

Залишилося налаштувати пріоритети переривань і маскувати їх для ініціації обробки. Зверніть увагу, що регістри NVIC_IPR доступні для побайтового звернення, що значно спрощує доступ тільки до необхідних байтам пріоритетів окремих векторів переривань. Досить тільки зробити зрушення на величину номера вектора переривання (див. Лістинг визначень). Ще раз нагадаємо, що EXTI Line 0 має 6 номер в таблиці векторів, а EXTI line 5_9 - номер 23. У STM32 значення мають тільки старші 4 біти пріоритету:

```
//Set interrupts priority
NVIC_IPR6_REG = 0xF0;
NVIC_IPR23_REG = 0x0;
```

Для демонстрації пріоритети встановлені різними.

Тепер можна включити переривання:

```
//Enable interrupts
NVIC_ISER0_REG |= NVIC_ISER0_6VECT | NVIC_ISER0_23VECT;
```

З цього моменту натискання на кнопку і закоротки PC6 і GND буде приводити до виклику функцій обробників переривань EXTI_Line0_IntHandler і EXTI_Line6_IntHandler відповідно.

16. Обробка переривань

У функціях обробки переривань в першу чергу необхідно очистити переривання, після цього можна запалити світлодіоди. Для демонстрації пріоритетів переривань в один з обробників доданий вічний цикл. Якщо

пріоритет переривання з вічним циклом нижче пріоритету другого - то воно не зможе бути викликано. Інакше, воно зможе перервати перше. Я пропоную вам самим спробувати різні значення пріоритетів переривань і наочно побачити до чого це призводить (УВАГА - не забудьте про групи переривань!).

```
    void EXTI_Line0_IntHandler(void)
    {
    //Clear interrupt
      EXTI_PR = EXTI_LINE0;
    //Turn on LED 3
      GPIOD_ODR |= GPIO_ODR_13PIN;
    }
    void EXTI_Line6_IntHandler(void)
    {
    //Clear interrupt
      EXTI_PR = EXTI_LINE6;
    //Turn LED4
      GPIOD_ODR |= GPIO_ODR_14PIN;
    while(1);
}
```

Тема 3.7 Аналого-цифровий та цифро-аналоговий перетворювачі

Всі пристрої **STM32** мають три 12-бітні АЦП та два ЦАП.

ЦАП

Основні характеристики:

- Вихідна напруга від 0 до VddA=2,9В – (напруги живлення аналогових компонентів)
- Аппаратна генерація шуму та сигналу трикутної форми
- Два незалежних канали
- Можливість роботи в 8 та 12 розрядному режимі з лівим чи правим вирівнюваням
- Зовнішні події: старт, зупин, ініціализація лічильника чи його оновлення внутрішнім чи зовнішнім запуском.
- Перетворений аналоговий сигнал (напругу) можна знімати з виводів PA5 та PA4

Регістр керування DAC_CR має вигляд

Таблиця 7.1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED		DMAUDRIE2	DMAEN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]			TEN2	BOFF2	EN2
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		DMAUDRIE1	DMAEN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

На ній зображені:

- ENx - запуск ЦАП. Деякі біти данного регістру неможливо змінити поки ENx встановлено одиницю. Тому рекомендовано встановлювати його в останю чергу, коли всі інші вже настроєні.
- BOFFx - включення буферу, за рахунок якого напруга на виході ЦАП не просідає під великим навантаженням.
- TENx – включення зовнішнього запуску перетворення.
- TSELx[2:0] – джерело запуску перетворення згідно табл. 7.2.
- DMAx – дозвіл переривання.

Таблиця 7.2

TSELx2	TSELx1	TSELx0	Джерело запуска
0	0	0	Подія TRGO таймера 6
0	0	1	Подія TRGO таймера 3
0	1	0	Подія TRGO таймера 6
0	1	1	Подія TRGO таймера 7
1	0	0	Подія TRGO таймера 5 чи 15
1	0	1	Подія TRGO таймера 4
1	1	0	Зовнішня лінія 9
1	1	1	Програмний запуск

- ЦАП може почати перетворення від сигналу одного з таймерів, в залежності від зміни логічного рівня на одному з виводів контролера (лінія 9) чи встановлення біта SWTRIGx у регістр DAC_SWTRIG (програмний запуск).
- **WAVEx[1:0]** – цими бітами можна включити генерацію білого шуму чи сигналу трикутної форми. Можливі наступні комбінації:

Таблиця 7.3

WAVEx1	WAVEx0	Вихідний сигнал
0	0	Генерація відсутня
0	1	Білий шум
1	X	Сигнал трикутної форми

• **MAMPx[3:0]** – біти задають амплітуду білого шуму чи сигналу трикутної форми(в залежності від того, що обрано бітами **WAVEx[1:0]**). Напруга на виході визначається наступним чином:

ЦАП = Vref*DORx/4095,

де Vref - напруга на одноіменому виводі, **DORx** значення відповідного регістру.

Електричні характеристики ЦАП наведено в табл. 7.4

Таблиця 7.4

Символ	Параметр	Min	Typ	Max	Unit
V_{DDA}	Аналогова напруга живлення	1.8(1)	-	3.6	V
V_{REF+}	Опорна напруга живлення	1.8(1)	-	3..6	V
V_{SSA}	Земля	0	-	0	V
R_{LOAD}(2)	Активне навантаження з буфером	5	-	-	kΩ
R_O(2)	Опір визоду з буфером	-	-	15	kΩ
G_{LOAD}(2)	Ємнісне навантаження	-	-	50	pF
DAC_OUT min(2)	Мінімальна напруга ЦАП з буфером	0.2	-	-	V
DAC_OUT max(2)	Максимальна напруга ЦАП з буфером		-	V_{DDA}-0.2	V

Приклад програми ЦАП

Розглянемо код який генерує сигнал трьох форм: білий шум, трикутну і синусоїдальну форму сигналу. Для генерації останнього - був використаний табличний метод.

```
#include "stm32f10x.h"
02.#include "stm32f10x_rcc.h"
```

```
03.#include "stm32f10x_gpio.h"
04.
05 /* Масив значень синусоїдального сигналу*/
06.const uint16_t sin[32] = {
07.2047, 2447, 2831, 3185, 3498, 3750, 3939, 4056, 4095, 4056,
08.3939, 3750, 3495, 3185, 2831, 2447, 2047, 1647, 1263, 909,
09.599, 344, 155, 38, 0, 38, 155, 344, 599, 909, 1263, 1647};
10.unsigned char i=0;
11.
12.int main(void) {
13./* ВмикаємопортА */
14.RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
15./* ВмикаємоЦАП */
16.RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
17./* Вмикаємотаймер 6 */
18.RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6,ENABLE);
19.
20./* налаштуваннявиводуЦАП */
21.GPIO_InitTypeDef GPIO_InitStructure;
22.GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
23.GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
24.GPIO_Init(GPIOA, &GPIO_InitStructure);
25.
26./* налаштуваннятаймеру*/
27.TIM6->PSC = 0;
28.TIM6->ARR = 500;
29.TIM6->DIER |= TIM_DIER_UIE; // дозвілпереривиннявідтаймеру
30.TIM6->CR1 |= TIM_CR1_CEN; // Почативідлік!
31.NVIC_EnableIRQ(TIM6_DAC_IRQn); // дозвілпереривання
TIM6_DAC_IRQn
32.
33./* Вмикаємо DAC1 */
34.DAC->CR |= DAC_CR_EN1;
35.
36./* Нескінченийцикл */
37.while (1)
38.{
39.}
40.}
41.
42./*Обробникпереривиннявідтаймеру 6 */
43.void TIM6_DAC_IRQHandler(void) {
```

44.TIM6->SR &= ~TIM_SR_UIF; //скиданняпрапору UIF
45.DAC->DHR12R1=sin[i++]; //Запис в ЦАП чергового елемент масиву
46.if (i==32) i=0; //Якщо вивели в ЦАП всі 32 значення то починаємо заново

Після запуску цієї програми на виводі PA4 можна зняти наступну осцилограму:

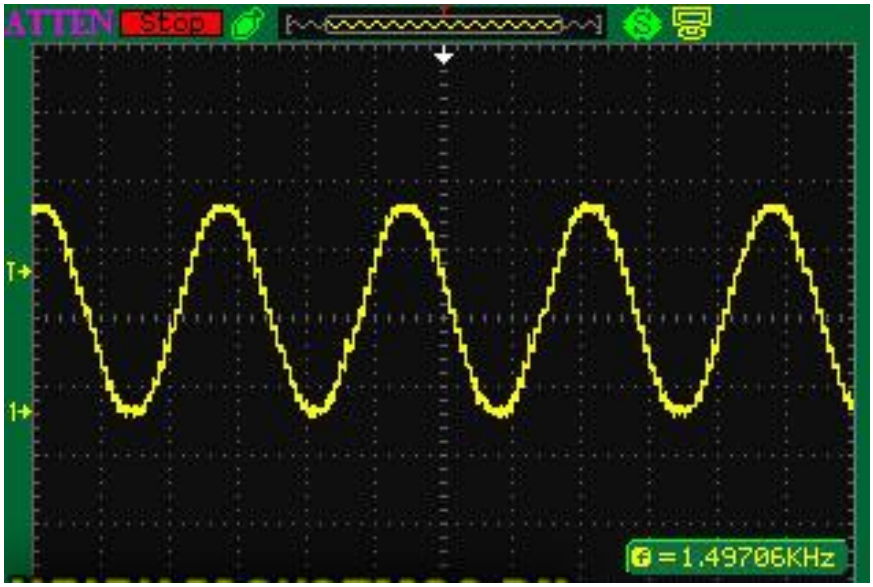


Рис.7.1

Розглянемо програму як можна отримати білий шум. Білий шум буде генеруватися апаратно. Для цього потрібно включити зовнішній запуск перетворення, а джерелом його запуску поставити таймер. Таймер буде дораховувати до значення записаного в регістрі ARR, а потім буде перезавантажуватиметеся і буде генерувати подію, яка запустить перетворення. І так нескінченно. В результаті на виході матимемо білий шум.

```
01.#include "stm32f10x.h"
02.#include "stm32f10x_rcc.h"
03.#include "stm32f10x_gpio.h"
04.
05.int main(void) {
06./* ВмикаємопортА */
07.RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
08./* ВмикаємоЦАП */
09.RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
10./* Вмикаємотаймер 6 */
11.RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6,ENABLE);
```

```
12.
13./* налаштуваннявиводуЦАПа */
14.GPIO_InitTypeDef GPIO_InitStructure;
15.GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
16.GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
17.GPIO_Init(GPIOA, &GPIO_InitStructure);
18.
19./* налаштуваннятаймер*/
20.TIM6->PSC = 0;
21.TIM6->ARR = 500;
22.TIM6->CR2=TIM_CR2_MMS_1; /* ТаймербудеджереломподійдляЦАПа*/
23.TIM6->CR1 |= TIM_CR1_CEN; // Почативідлік!
24.
25./* Вмикаємо DAC1 */
26.DAC->CR |= DAC_CR_TEN1; /* Перетворенняповиникненнюподії... */
27.DAC->CR &= ~DAC_CR_TSEL1; /* ... відтаймеру 6*/
28.DAC->CR |= DAC_CR_WAVE1_0; /* Генераціяшуму */
29.//DAC->CR |= DAC_CR_WAVE1_1; /* Генераціясигналутрикутноїформи */
30.DAC->CR |= DAC_CR_MAMP1; /* Максимальнаамплітуда */
31.DAC->CR |= DAC_CR_EN1; /* ВімкнутиЦАП1 */
32.
33./* Нескінчений цикл */
34.while (1)
35.{
36.}
37.}
```

На екрані осцилографа можна отримати наступну осцилограму:

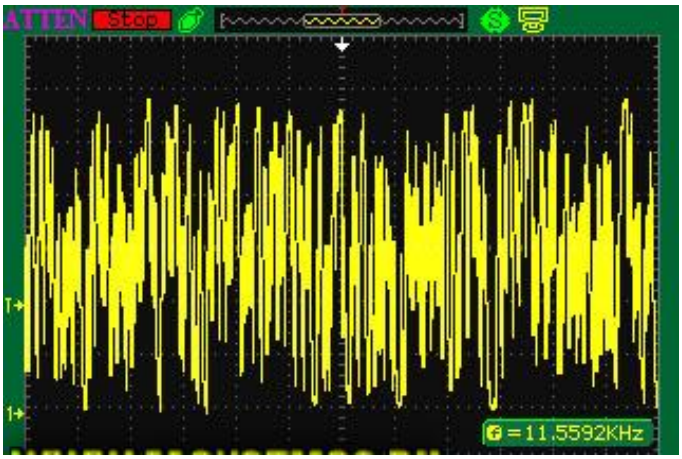


Рис.7.2

ЦАП можна використовувати як генератор псевдовипадкових чисел.

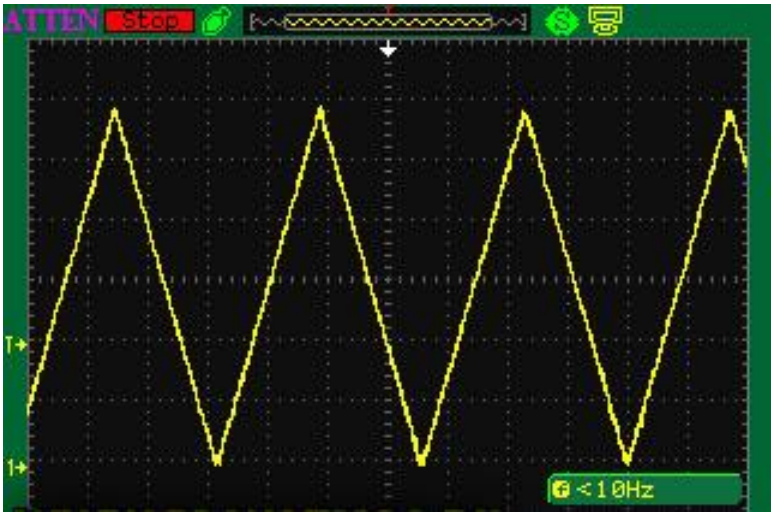


Рис.7.3

Якщо потрібно генерувати трикутник, то код не зміниться за винятком того, що потрібно за коментувати строку номер 28, а строку 29 включити.

Таблиця 7.5

Функція ЦАП	Опис
void DAC_DeInit(void);	Функції конфігурації каналів ЦАП: запуску, вихідного буфера, Формату даних
void DAC_Init(uint32_t DAC_Channel, DAC_InitTypeDef* DAC_InitStruct);	
void DAC_StructInit(DAC_InitTypeDef* DAC_InitStruct);	
void DAC_Cmd(uint32_t DAC_Channel, FunctionalState NewState);	
void DAC_SoftwareTriggerCmd(uint32_t DAC_Channel, FunctionalState NewState);	
void DAC_DualSoftwareTriggerCmd(FunctionalState NewState);	
void DAC_WaveGenerationCmd(uint32_t DAC_Channel, uint32_t DAC_Wave, FunctionalState NewState);	
void DAC_SetChannel1Data(uint32_t DAC_Align, uint16_t Data);	
void DAC_SetChannel2Data(uint32_t DAC_Align, uint16_t Data);	
void DAC_SetDualChannelData(uint32_t DAC_Align, uint16_t Data2, uint16_t Data1);	

uint16_t DAC_GetDataOutputValue(uint32_t DAC_Channel);	
void DAC_DMACmd(uint32_t DAC_Channel, FunctionalState NewState);	Функції управління DMA
void DAC_ITConfig(uint32_t DAC_Channel, uint32_t DAC_IT, FunctionalState NewState);	Функції управління перериваннями і прапорами
FlagStatus DAC_GetFlagStatus(uint32_t DAC_Channel, uint32_t DAC_FLAG);	
void DAC_ClearFlag(uint32_t DAC_Channel, uint32_t DAC_FLAG);	
ITStatus DAC_GetITStatus(uint32_t DAC_Channel, uint32_t DAC_IT);	
void DAC_ClearITPendingBit(uint32_t DAC_Channel, uint32_t DAC_IT);	

Опис структури

```
typedef struct
 {
  uint32_t DAC_Trigger;
  uint32_t DAC_WaveGeneration;
  uint32_t DAC_LFSRUnmask_TriangleAmplitude;
uint32_t DAC_OutputBuffer;
}DAC_InitTypeDef;
```

DAC_Trigger (зовнішній тригер для обраного каналу ЦАП)

```
DAC_Trigger_T6_TRGO
DAC_Trigger_T8_TRGO
DAC_Trigger_T7_TRGO
DAC_Trigger_T5_TRGO
```

DAC_Trigger_T2_TRGO

DAC_Trigger_T4_TRGO

DAC_Trigger_Ext_IT9

DAC_Trigger_Software

DAC_WaveGeneration (ВказуєнаформуканалуЦАП (спеціальна (шум, трикутник) або ні)

DAC_WaveGeneration_None

DAC_WaveGeneration_Noise

DAC_WaveGeneration_Triangle

DAC_LFSRUnmask_TriangleAmplitude pecifies (маска для генерації шуму або максимальна амплітуда трикутника)

DAC_LFSRUnmask_Bit0

DAC_LFSRUnmask_Bits1_0

…

DAC_LFSRUnmask_Bits11_0 (Unmask DAC channel LFSR bit[11:0] for noise wave generatio)

DAC_TriangleAmplitude_1

…

DAC_TriangleAmplitude_4095

DAC_OutputBuffer (дозвіл/заборонавиходуканалаЦАП)

DAC_OutputBuffer_Enable

DAC_OutputBuffer_Disabl

1. АЦП

Мікроконтролер STM32F407VG включає в себе три АЦП. Розрядність всіх АЦП становить 12 біт.

Кожен перетворювач здатний приймати сигнал з шістнадцяти зовнішніх каналів. Крім того, до складу контролера входить датчик температури. Діапазон вхідних напруг становить 1.8 ... 3.6 В. Датчик температури підключений до вхідного каналу ADC_IN16, який використовується для того, щоб перетворити вихідну напругу сенсора в цифрове значення.

Внутрішній датчик температури призначений для відстеження зміни температури, а не для її вимірювання, оскільки зсув показників датчика може змінюватися в ході змін параметрів процесу. Тому, якщо необхідно точне вимірювання абсолютних значень температури, то для цього краще використовувати зовнішній датчик.

До складу АЦП входить аналогова схема контролю (**Analog Watchdog**), яка потрібна для того, щоб стежити, що напруга потрапляє в певні межі. Причому АЦП може сканувати як конкретний канал, так і групу каналів. В регістри ADC_HTR і ADC_LTR заносяться значення верхнього та нижнього порога відповідно. І у випадку, якщо напруга виходить за ці межі, генерується переривання. Як і в мікроконтролерах AVR можливо вирівнювання результату по правому або по лівому краю. Результат перетворення - 12-бітний.

Основні характеристики АЦП:

- АЦП є 12-бітовим
- Можлива генерація переривання після закінчення перетворення, по закінченню перетворення з інжектованного каналу, а також можливе переривання від Analog Watchdog

Канали АЦП мікроконтролерів STM32 діляться на дві групи: регулярні канали (regular) і інжектовані (injected). Кількість регулярних каналів для одного АЦП дорівнює 18, серед них 16 зовнішніх і два внутрішніх - опорна напруга і температурний датчик. Кількість пріоритетних каналів дорівнює

чотирьом. Інжектовані канали мають власні регістри для зберігання результату, а регулярні канали для збереження результатів використовують DMA (прямий доступ до пам'яті). Можливе одиничне перетворення і перетворення в безперервному режимі.

Особливості:

- Самокалібровка
- Запуск перетворення від зовнішньої події
- Робота з ПДП (прямий доступ до пам'яті)

Електричні характеристики АЦП наведені в табл. 7.6

Таблиця 7.6

Символ	Параметр	Умови	Min	Typ	Max	Unit
V_{DDA}	Напруга живлення		1.8(1)	-	3.6	V
V_{REF+}	Полюс опорної напруги		1.8(1)(2)(3)	-	V_{DDA}	V
f_{ADC}	Тактова частота АЦП	V_{DDA} = 1.8(1)(3) to 2.4 V	0.6	15	18	MHz
		V_{DDA} =2.4to 3.6V(3)	0.6	30	36	MHz
f_{TRIG}(4)	Частота зовнішнього тригера	f_{ADC} = 30 NHz, 12-bit resolution	-	-	1764	kHz
			-	-	17	$1/f_{ADC}$
V_{AIN}	Діапазон перетворення напруг		0 (V_{SSA} or V_{REF} – tied to ground)	-	V_{REF+}	V
R_{AIN}(4)	Зовнішній вхідний опір	See Equation 1 for details	-	-	50	kΩ
R_{AIN}(4)(6)	Вхідний опір АЦП		-	-	6	kΩ

C_{ADC}[4]	Ємність входу АЦП		-	4	-	pF
t_{lat}[4]	Затримка перетворення	f_{ADC} = 30 NHz	-	-	0.100	µs
			-	-	3[7]	$1/f_{ADC}$
t_{latr}[4]	Затримка перетворення	f_{ADC} = 30 NHz	-	-	0.067	µs
			-	-	2[7]	$1/f_{ADC}$
t_s[4]	Затримка перетворення	f_{ADC} = 30 NHz	0.100	-	16	µs
			3	-	480	$1/f_{ADC}$
t_{STAB}[4]	Час стабілізації		-	2	3	µs

2. Регістри АЦП

У випадку програмування вхідних каналів АЦП як регулярних, результат вимірювання записується в регістр **ADC_DR**:

Таблиця 7.7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

У цей регістр записуються дані, отримані в результаті опитування каналів, що складаються в деякій регулярній групі. Кожний наступний вимір одного каналу стирає попереднє. Для того щоб своєчасно зчитувати дані і записувати їх в певні змінні потрібно використовувати переривання або DMA. Таким чином, якщо не використовується переривання DMA, то використовувати регулярні групи можна тільки якщо канал в цій групі всього один. Для зручності дані можуть вирівнюватися як по лівому краю регістра так і по правому. Це здійснюється бітом ALIGN регістра ADC_CR2.

У випадку програмування вхідних каналів як інжектованих регістри ADC_JDRx записуються результати перетворення для кожного з 4-х

інжектованих каналів. Регулярна група каналів може бути перервана почерговим запуском інжектованих каналів (рис.7.4). При завершенні перетворення групи зупиняються всі інші перетворення обох груп, і результати обробки зберігаються в регістрах даних кожного АЦП.

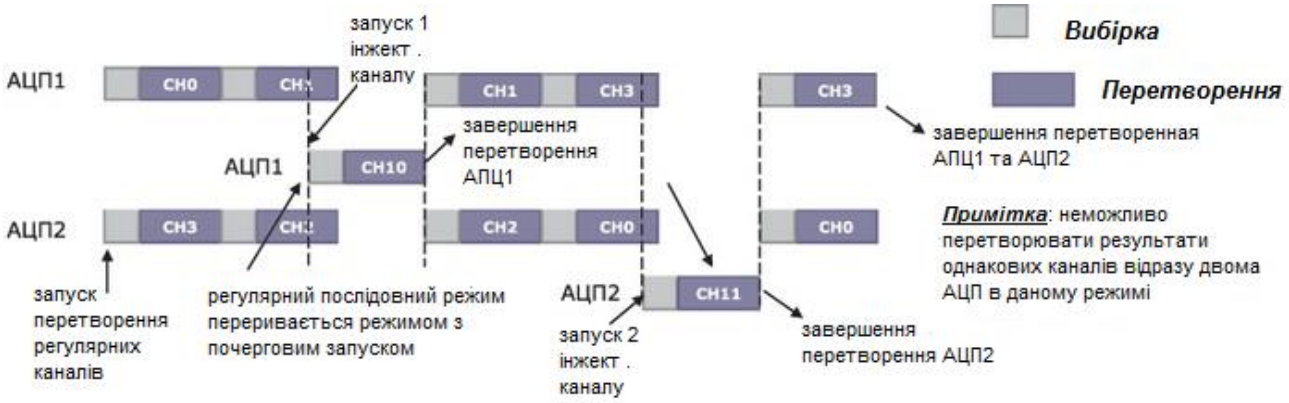


Рис. 7.4

Регістр **ADC_SR** відображає стан АЦП на даний момент:

Таблиця 7.8

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										OVR	STRT	JSTRT	JEOC	EOC	AWD
										rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

STRT - запуск перетворення для регулярного каналу

JSTRT - запуск перетворення для інжектованного каналу

JEOC - кінець перетворення для інжектованного каналу

EOC - кінець перетворення інжектованного або регулярного каналу

AWD - Analog watchdog спрацював

Всі біти скидаються програмним шляхом.

Для налаштування основних АЦП використовують два регістри **ADC_CR1** та **ADC_CR2**. Структура першого з них виглядає наступним чином:

Таблиця 7.9

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED								AWDEN	JAWDEN	RESERVED					
								rw	rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

AWDEN - Analog watchdog моніторить стан всіх каналів регулярної групи.

JAWDEN - Analog watchdog моніторить стан всіх каналів інжектованої групи.

AWDSGL - Analog watchdog моніторить стан одного каналу зазначеного в бітах **AWDCH.**

AWDCH[4:0] - Номер каналу АЦП який буде моніторити Analog watchdog.

В залежності від ситуації, біти встановлюються наступним чином:

Таблиця 7.10

Канали під моніторингом **Analogwatchdog**	Біти регістра **ADC_CR1**		
	AWDSGL	**AWDEN**	**JAWDEN**
Жодний канал	x	0	0
Всі інжектовані	0	0	1
Всі регулярні	0	1	0
Всі інжектовані та регулярні	0	1	1
Один інжектований канал	1	0	1
Один регулярний канал	1	1	0
Один регулярний чи інжектований канал	1	1	1

JAUTO - дозволяє автоматичне перетворення для каналів в інжектовані групі після каналів регулярної групи.

SCAN – встановлюється при одночасному опитувані декількох каналів АЦ.

JEOCIE - вмикає або вимикає переривання для інжектованих каналів
AWDIE - вмикає або вимикає переривання від Analog watchdog
EOCIE - вмикає або вимикає переривання після закінчення перетворення в регулярній або інжектовані групі

Регістр **ADC_CR2** має вигляд

Таблиця 7.11

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTEL[2:0]			Res.
								rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTTRIG	JEXTSEL[2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
rw	rw	rw	rw	rw	Res.		rw					rw	rw	rw	rw

TSVREFE - включає температурний сенсор. До установки цього біта читати щось з каналу до якого він підключений - марно.

SWSTART - запускає перетворення для каналів входять до складу регулярної групи. Для того, щоб перетворення було запущено цим бітом потрібно попередньо встановити біти EXTSEL [2: 0] в одиниці.

JSWSTART - запускає перетворення для каналів що входять до складу інжектованої групи. Для того, щоб перетворення було запущено цим бітом потрібно попередньо встановити біти JEXTSEL [2: 0] в одиниці.

EXTTRIG - дозволяє використовувати зовнішню подію для старту перетворення каналів в регулярній групі.

EXTSEL[2:0] - цими бітами вибирається джерело яке буде запускати перетворення каналів в регулярній групі.

Доступні наступні бітові комбінації табл. 7.12:

Таблиця 7.12

EXTSEL2	EXTSEL1	EXTSEL0	Джерело запуску

0	0	0	Подія CC1 таймера 1
0	0	1	Подія CC2 таймера 1
0	1	0	Подія CC3 таймера 1
0	1	1	Подія CC2 таймера 2
1	0	0	Подія TRGO таймера 3
1	0	1	Подія CC4 таймера 4
1	1	0	Зовнішня лінія 11
1	1	1	Програмний старт

JEXTTRIG - дозволяє використовувати зовнішню подію для старту перетворення каналів в інжектованній групі.

JEXTSEL[2:0] - цими бітами вибирається джерело яке буде запускати перетворення каналів в інжектованній групі. Доступні наступні бітові комбінації наведено табл. 7.13:

Таблиця 7.13

JEXTSEL2	JEXTSEL1	JEXTSEL0	Джерело
0	0	0	Подія TRGO таймера1
0	0	1	Подія CC4 таймера 1
0	1	0	Подія TRGO таймера 2
0	1	1	Подія CC1 таймера 2
1	0	0	Подія CC4 таймера 3
1	0	1	Подія TRGO таймера 4
1	1	0	Зовнішня лінія 15
1	1	1	Програмний старт

ALIGN - встановлює вирівнювання результату в регістрі даних (0 - по правому, 1 - по лівому)

RSTCAL - скидає значення калібрування

CAL - запускає калібрування АЦП. Після завершення калібрування біт встановлюється в нуль

CONT - дозволяє безперервне перетворення

ADON - включає / вимикає модуль АЦП

DMA - дозволяє використовувати DMA

Регістри **ADC_SMPR1** і **ADC_SMPR2.** У них задається час вибірки індивідуально для кожного каналу.

Таблиця 7.14

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED								SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED		SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]			SMP5[2:1]	
RES.		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP5_0	SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]			SMP0[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Час вибірки визначається числом тактів ADCCLK, задається в розрядах SMPx [2: 0] регістрів ADC_SMPRx індивідуально для кожного канала.

Чотире регістра **ADC_JOFRx** зберігають значення які будуть відраховуватися з відповідних регістрів **ADC_JDRx.**

Таблиця 7.15

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				JOFFSETx[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Регістри **ADC_HTR** і **ADC_LTR** задають верхню і нижню межу з якими Analog watchdog порівнює значення обраного каналу АЦП.

Регістр **ADC_SQRx** визначають, які канали входять в регулярну групу. Всього можна додати не більше 16-ти каналів. Бітами L [3: 0] потрібно задати число каналів у групі (0000 - 1 канал ... 1111 - 16 каналів).

Таблиця 7.16

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED								L[3:0]				SQ16[4:1]			
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]					SQ14[4:0]					SQ13[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED		SQ12[4:0]					SQ11[4:0]					SQ10[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ10_0	SQ9[4:0]					SQ8[4:0]					SQ7[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED		SQ6[4:0]					SQ5[4:0]					SQ4[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4_0	SQ3[4:0]					SQ2[4:0]					SQ1[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Регістр **ADC_JSQR** задає номери каналів, які входять до складу інжектованої групи :

Таблиця 7.17

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED										JL[1:0]		JSQ4[4:1]			
										rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSQ4_0	JSQ3[4:0]					JSQ2[4:0]					JSQ1[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Бітами **JSQx** задаються номери каналів, а в JL [1: 0] записується кількість каналів у інжектованій групі. Якщо в інжектовану групу додані всі чотири канали то, в регістр **ADC_JDR1** будуть записувати дані з каналу зазначеного бітами **JSQ1**, в **ADC_JDR2** з каналу **JSQ2** і т.д. Якщо число каналів менше 4, то їх вибір визначається табл. 7.18.

Таблиця 7.18

Кількість каналів	ADC_JDR1	ADC_JDR2	ADC_JDR3	ADC_JDR4
4	JSQ1	JSQ2	JSQ3	JSQ4
3	JSQ2	JSQ3	JSQ4	
2	JSQ3	JSQ4		
1	JSQ4			

З таблиці видно, що якщо включити в групу тільки один канал необхідно вибрати біт **JSQ4**.

3. Приклад програми АЦП

Розглянемо приклад який зчитує значення каналу АЦП, а потім виходячи з цього значення будемо міняти частоту мигання світлодіодів STM32 Discovery.

У даному прикладі будемо використовувати інжектовані канали з міркувань простоти і наочності. Для того щоб прочитати дані з каналу ADC1 необхідно зробити наступне:

1) Включити тактування порту A

2) Налаштувати вивід PA1 як вхід без підтяжки (за замовчуванням вона вже в такому стані)

3) Включити тактування АЦП

4) Почати калібрування і дочекатися його завершення

5) Додати канал ADC1 до складу інжектованих груп

6) Вибрати джерелом запуску біт JSWSTART

7) Активувати режим безперервного перетворення

8) Включити АЦП

9) Запустити перетворення

10) Дочекатися завершення першого перетворення

11) Прочитати результат

```
01.#include<stm32f10x_rcc.h>
02.#include<stm32f10x_gpio.h>
03.#include "stm32f10x.h"
04.void delay(uint32_t i) {
05.volatile uint32_t j;
06.for (j=0; j!= i * 1000; j++)
07.;
08.}
09.
10.int main(void)
11.{
12.GPIO_InitTypeDef PORT;
13.//ВмикаємопортиАиС
14.RCC_APB2PeriphClockCmd((RCC_APB2Periph_GPIOC |
RCC_APB2Periph_GPIOA) , ENABLE);
15.//Налагоджуємо PC8 і PC9 на вихід. Світлодіоди
```

```
16.PORT.GPIO_Pin = (GPIO_Pin_9 | GPIO_Pin_8);
17.PORT.GPIO_Mode = GPIO_Mode_Out_PP;
18.PORT.GPIO_Speed = GPIO_Speed_2MHz;
19.GPIO_Init( GPIOC , &PORT);
20./Порт налагоджено за замовчанням
21.RCC_APB2PeriphClockCmd(RCC_APB2ENR_ADC1EN, ENABLE); //Вмикаємо тактування АЦП
22.ADC1->CR2 |= ADC_CR2_CAL; //Запуск калібровки АЦП
23.while (!(ADC1->CR2 & ADC_CR2_CAL))
24.; //чекаємозакінченнякалібровки
25.ADC1->SMPR2 |= (ADC_SMPR2_SMP1_2 | ADC_SMPR2_SMP1_1 | ADC_SMPR2_SMP1_0); //Задаєм
26.// тривалістьвибірки
27.ADC1->CR2 |= ADC_CR2_JEXTSEL; //Перетворенняінжектованноїгрупи
28.//запуститьсяустановкоюбіта JSWSTART
29.ADC1->CR2 |= ADC_CR2_JEXTTRIG; //дозволитизовнішнійзапускінжектованноїгрупи
30.ADC1->CR2 |= ADC_CR2_CONT; //Перетворення запускаються одне за другим
31.ADC1->CR1 |= ADC_CR1_JAUTO; // дозволити перетворення інжектованної групи
33.ADC1->JSQR |= (1<<15); //Задаємо номер каналу (обраний ADC1)
34.ADC1->CR2 |= ADC_CR2_ADON;//Тепер вмикаємо АЦП
35.ADC1->CR2 |= ADC_CR2_JSWSTART; //Запуск перетворення
36.while (!(ADC1->SR & ADC_SR_JEOC)) // перше перетворення
37.;
38.//Прочитати результат з JDR1
39.uint32_t adc_res; //Використовувати змінну для налагодження.
40.while(1)
41.{
42.adc_res=ADC1->JDR1;
43.delay(adc_res); //ВиходячиіззначенняАЦПробимозатримку
44.GPIO_WriteBit(GPIOC,GPIO_Pin_8,Bit_RESET); //горитьдіод...
45.GPIO_WriteBit(GPIOC,GPIO_Pin_9,Bit_SET); // Негоритьдіод
46.adc_res=ADC1->JDR1;
47.delay(adc_res); //
48.GPIO_WriteBit(GPIOC,GPIO_Pin_9,Bit_RESET);
49.GPIO_WriteBit(GPIOC,GPIO_Pin_8,Bit_SET);
50.}
51.}
```

Калібрування треба запускати до включення АЦП установкою біта **ADC_CR2_ADON**. Алгоритм наступний: Налаштовуємо порти і АЦП, в нескінченному циклі забираємо результат перетворення і передаємо його в функцію затримки. Таким чином затримка буде залежати від напруги на PA1.

Таблиця 7.19

Функція АЦП	**Опис**

void ADC_DeInit(void);	Функція конфігурації за замовчанням
void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct);	Функція ініціалізації і конфігурації
void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct);	
void ADC_CommonInit(ADC_CommonInitTypeDef* ADC_CommonInitStruct);	
void ADC_CommonStructInit(ADC_CommonInitTypeDef* ADC_CommonInitStruct);	
void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState);	
void ADC_AnalogWatchdogCmd(ADC_TypeDef* ADCx, uint32_t ADC_AnalogWatchdog);	Функціяконфігурації аналоговоїсхеми контролю (Analog Watchdog)
void ADC_AnalogWatchdogThresholdsConfig(ADC_TypeDef* ADCx, uint16_t HighThreshold,uint16_t LowThreshold);	
void ADC_AnalogWatchdogSingleChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel);	
void ADC_TempSensorVrefintCmd(FunctionalState NewState);	Функціїкерування датчиктемператури , Vrefint і VBAT
void ADC_VBATCmd(FunctionalState NewState);	
void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime);	Функції конфігурації регулярних каналів АЦП
void ADC_SoftwareStartConv(ADC_TypeDef* ADCx);	
FlagStatus ADC_GetSoftwareStartConvStatus(ADC_TypeDef* ADCx);	
void ADC_EOCOnEachRegularChannelCmd(ADC_TypeDef* ADCx, FunctionalState NewState);	
void ADC_ContinuousModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState);	
void ADC_DiscModeChannelCountConfig(ADC_TypeDef* ADCx, uint8_t Number);	
void ADC_DiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState);	
uint16_t ADC_GetConversionValue(ADC_TypeDef* ADCx);	
uint32_t ADC_GetMultiModeConversionValue(void);	
void ADC_DMACmd(ADC_TypeDef* ADCx, FunctionalState NewState);	Функції конфігурації DMA регулярних каналів
void ADC_DMARequestAfterLastTransferCmd(ADC_TypeDef* ADCx, FunctionalState NewState);	
void ADC_MultiModeDMARequestAfterLastTransferCmd(FunctionalState NewState);	
void ADC_InjectedChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime);	Функції конфігурації інжектованих каналів АЦП
void ADC_InjectedSequencerLengthConfig(ADC_TypeDef* ADCx, uint8_t Length);	

void ADC_SetInjectedOffset(ADC_TypeDef* ADCx, uint8_t ADC_InjectedChannel, uint16_t Offset);	
void ADC_ExternalTrigInjectedConvConfig(ADC_TypeDef* ADCx, uint32_t ADC_ExternalTrigInjecConv);	
void ADC_ExternalTrigInjectedConvEdgeConfig(ADC_TypeDef* ADCx, uint32_t ADC_ExternalTrigInjecConvEdge);	
void ADC_SoftwareStartInjectedConv(ADC_TypeDef* ADCx);	
FlagStatus ADC_GetSoftwareStartInjectedConvCmdStatus(ADC_TypeDef* ADCx);	
void ADC_AutoInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState);	
void ADC_InjectedDiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState);	
uint16_t ADC_GetInjectedConversionValue(ADC_TypeDef* ADCx, uint8_t ADC_InjectedChannel);	
void ADC_ITConfig(ADC_TypeDef* ADCx, uint16_t ADC_IT, FunctionalState NewState);	Функції управління перериваннями і прапорами
FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint8_t ADC_FLAG);	
void ADC_ClearFlag(ADC_TypeDef* ADCx, uint8_t ADC_FLAG);	
ITStatus ADC_GetITStatus(ADC_TypeDef* ADCx, uint16_t ADC_IT);	
void ADC_ClearITPendingBit(ADC_TypeDef* ADCx, uint16_t ADC_IT);	

typed
efstruct
{
uint32_t ADC_Resolution; Налаштовує режим подвійної роздільної здатності
FunctionalState ADC_ScanConvMode; включення багатоканального/однокального режиму (ENABLE or DISABLE)
FunctionalState ADC_ContinuousConvMode; безперевне/одиночнеперетворення(ENABLE or DISABLE)
uint32_t ADC_ExternalTrigConvEdge; Вибірфронтузапускурегулярноїгрупи
uint32_t ADC_ExternalTrigConv; Вибірзовнішньоїподіїдлязапускуперетвореннярегулярноїгрупи
uint32_t ADC_DataAlign; ВирівнюваннянданихвАЦПвлівоабовправо.
uint8_t ADC_NbrOfConversion; НомерканалуАЦПперетворення
}**ADC_InitTypeDef**;

ADC_Resolution
ADC_Resolution_12b
ADC_Resolution_10b
ADC_Resolution_8b
ADC_Resolution_6b

ADC_ExternalTrigConvEdge
ADC_ExternalTrigConvEdge_None
ADC_ExternalTrigConvEdge_Rising
ADC_ExternalTrigConvEdge_Falling
ADC_ExternalTrigConvEdge_RisingFalling

ADC_ExternalTrigConv
ADC_ExternalTrigConv _T1_CC1
ADC_ExternalTrigConv _T1_CC2
ADC_ExternalTrigConv _T1_CC3
ADC_ExternalTrigConv _T2_CC2
ADC_ExternalTrigConv _T2_CC3
ADC_ExternalTrigConv _T2_CC4
ADC_ExternalTrigConv _T2_TRGO
ADC_ExternalTrigConv _T3_CC1
ADC_ExternalTrigConv _T3_TRGO
ADC_ExternalTrigConv _T4_CC4
ADC_ExternalTrigConv _T5_CC1
ADC_ExternalTrigConv _T5_CC2
ADC_ExternalTrigConv _T5_CC3
ADC_ExternalTrigConv _T8_CC1
ADC_ExternalTrigConv _T8_TRGO
ADC_ExternalTrigConv_Ext_IT11

ADC_DataAlign

ADC_DataAlign_Right
ADC_DataAlign_Left

ADC_NbrOfConversion

```
type   {
defstruct
        uint32_t ADC_Mode;                режим : незалежний канал/група
      uint32_t ADC_Prescaler;             Вибір частоти тактового сигнала
      uint32_t ADC_DMAAccessMode;         Прямий доступ до пам'яті або
      мульти режим АЦП
      uint32_t ADC_TwoSamplingDelay;      затримка між 2 фазами вибірки.
      }ADC_CommonInitTypeDef;
```

ADC_Mode

ADC_Mode_Independent
ADC_DualMode_RegSimult_InjecSimult
ADC_DualMode_RegSimult_AlterTrig)
ADC_DualMode_InjecSimult
ADC_DualMode_RegSimult
ADC_DualMode_Interl
ADC_DualMode_AlterTrig
ADC_TripleMode_RegSimult_InjecSimult
ADC_TripleMode_RegSimult_AlterTrig
ADC_TripleMode_InjecSimult
ADC_TripleMode_RegSimult
ADC_TripleMode_Interl
ADC_TripleMode_AlterTrig

ADC_Prescaler
ADC_Prescaler_Div2
ADC_Prescaler_Div4
ADC_Prescaler_Div6
ADC_Prescaler_Div8

ADC_DMAAccessMode
ADC_DMAAccessMode_Disabled
ADC_DMAAccessMode_1
ADC_DMAAccessMode_2
ADC_DMAAccessMode_3

ADC_TwoSamplingDelay
ADC_TwoSamplingDelay_5Cycles
ADC_TwoSamplingDelay_6Cycles …
ADC_TwoSamplingDelay_20Cycles

Тема 3.8 I2C, SPI, USART, SDIO ТА SAI ІНТЕРФЕЙСИ.

1. Опис інтерфейсу I2C

Inter-IntegratedCircuit (I2C) модуль - послідовний інтерфейс, призначений, підтримки зв’язку з периферійними приладами мікроконтроллера. Периферійними приладами можуть бути послідовні EEPROM, драйвери дисплея, АЦП, тощо.

2. Характеристики шини I2C

Шина I2C - це двопровідний послідовний інтерфейс. На рис. 8.1 приведено схемна реалізація шини I2C, яка є типовим прикладом для будь-якого інтерфейсу I2C.

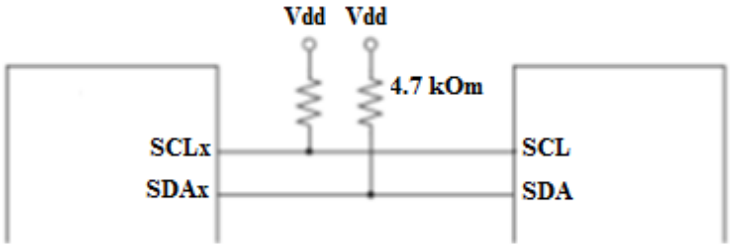


Рис. 8.1. Типова схема організації шини I2C

Інтерфейс використовує двонаправлений протокол, щоб гарантувати надійну передачу і прийом даних. При з'єднанні один пристрій виконує роль ведучого. Ведучий починає передачу даних на шині, а також генерує тактові імпульси А. Другий пристрій є веденим, що відповідає на передачу. Лінія тактування SCLx керується ведучим, і по ній тактові імпульси надходять веденому, хоча можливі ситуації, коли і ведений може керувати лінією SCLx. Лінія даних, SDAx, може бути, як виходом, так і входом, як для ведучого, так і для веденого.

Оскільки лінії SDAx і SCLx двонаправлені, то виводи пристроїв повинні мати лінії SDAx і SCLx з відкритим колектором, щоб забезпечити функцію

логічного I на шині. Зовнішні підтягуючі резистори використовуються, щоб гарантувати високий рівень на лініях, коли жоден з пристроїв не керує шиною.

У протоколі інтерфейсу I2C, кожен пристрій має свою власну адресу. Коли ведучий бажає ініціалізувати передачу даних, то він спочатку передає адресу пристрою, з яким потрібно встановити зв'язок. Нульовий біт адреси визначає, чи хоче ведучий прочитати або записати дані з / на ведений пристрій. Ведучий і ведений завжди знаходяться в протилежних режимах під час передачі даних:

• Ведучий-передатчик и Ведений-приймач

• Ведений-передатчик и Ведучий-приймач

3. Шинний протокол

Передача данних може бути розпочата, тільки коли шина не зайнята.Протягом передачі даних, стан на лінії даних НЕ повинен змінюватися в той час, коли на тактовій лінії SCLx знаходиться високий рівень. Оскільки цей режим зарезервований для службових потреб. Зокрема, таким чином формуються умови START і STOP.

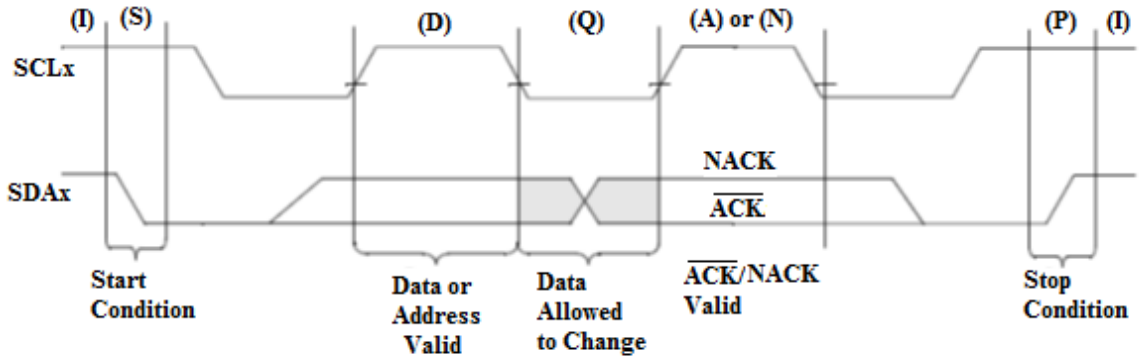


Рис. 8.2. Стан протоколу шини I2C

Умова START (S)

Після простою шини, коли лінії SCLx знаходиться на високому рівні, необхідно на лінії SDAx змінити рівень з високого на низький. Це і буде відповідати умові START. Перед тим як передавати дані, потрібно виконати умову START.

Умова STOP (P)

Перехід на лінії SDAx від низького до високого рівня, в той час як тактова лінія (SCLx) знаходиться у високому стані - відповідає умові STOP. Всі передачі даних повинні закінчуватися умовою STOP.

Умова повторного START (R)

Після неактивного стану шини, перехід на лінії SDAx від високого до низького рівня, в той час як тактова лінія (SCLx) знаходиться у високому стані - визначає умову повторного START. Повторний START дозволяє ведучому змінити режим керування шиною або адресу веденого без скидання передачі.

Достовірність даних (D)

Стан лінії SDAx надає достовірні дані, коли, після умови START, лінія SDAx стійка в той час, коли на тактовій лінії присутній високий рівень. Один такт на SCLx передає один біт даних.

Підтвердження (A), або не підтвердження (N)

Всі передачі байту даних повинні бути підтверджені (ACK) або не підтверджені (NACK) приймачем. Для формування ACK приймач встановлює на лінії SDAx низький рівень, або відпускає лінію SDAx умови NACK.

Затримка чи данні недійсні(Q)

Дані на лінії повинні змінюватися, коли на тактовій лінії присутній низький рівень. Пристрій також може затримати низький рівень на тактовій

лінії SCLx, що викликає затримку на шині (наприклад, для підготовки даних для відповіді без очищення шини).

Простій шини (I)

Це стан, коли на обох лініях присутній високий рівень. У цьому стані шина знаходиться до умови START і після умови STOP.

3. Протокол повідомлення

Типове I2C повідомлення показано на малюнку 3. У цьому прикладі, повідомлення буде читати зазначений байт з послідовної EEPROM 24LC256 по шині I2C.

Малюнок 3 показує дані на провідному устрої і на відомому пристрої, беручи до уваги, що об'єднана лінія SDAx включена за схемою логічного I між ведучим і веденим. Ведучий пристрій забезпечує роботу протоколом. Ведений пристрій буде керувати шиною тільки в певний час.

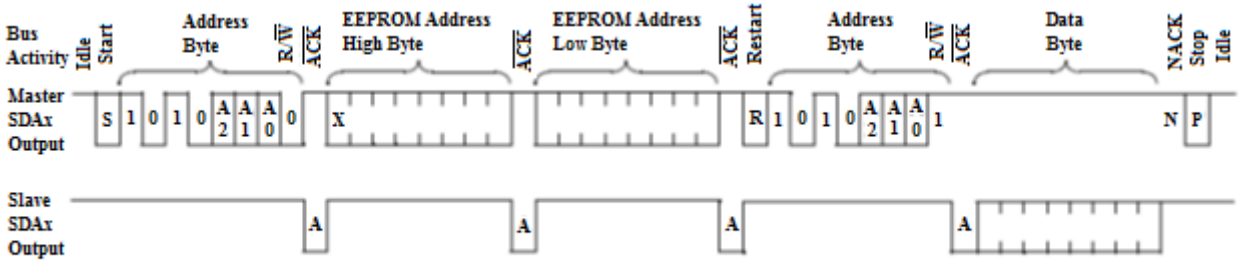


Рис 8.3. Повідомлення зчитування даних с EEPROM

Кожне повідомлення ініціалізується умовою START і закінчується умовою STOP. Число байт даних переданих між умовою START і умовою STOP - визначає ведучий пристрій.

На Рис. 8.3, перший байт - це байт адреси пристрою, який повинен завжди слідувати за першим байтом в повідомленні I2C. Воно містить адресу пристрою та стан біта R / W. Якщо R / W = 0, то це вказує на те, що для

адресованого веденого буде встановлений режим, коли ведучий буде передавачем, а ведений приймачем.

Приймач повинен генерувати підтвердження "ACK", після прийому кожного байту. Ведучий повинен генерувати додаткові імпульси SCLx, для підтверджень.

Наступні два байти, послані ведучим відомому, є байтами даних, що містять місце розташування необхідного байту даних EEPROM. Ведений повинен підтверджувати прийом кожного байту даних.

4. Повторний START

В даному місці, відоме EEPROM має адресу байту даних, яку необхідно повернути ведучому. Однак, біт R / W ще від першого байту повідомлення (адресного) визначає, що ведучий є передавачем, а ведений приймачем. Шина повинна бути перевизначена, щоб ведучий брав дані, а ведений відправляв дані. Щоб виконати цю функцію, не обриваючи повідомлення, провідний посилає "Повторний START".

Повторний START супроводжується байтом адресу пристрою, що містить той самий адрес пристрою, як і раніше, але з бітом R / W = 1, щоб вказати на те, що тепер ведучий буде приймати, а ведений передавати.

Тепер ведений передає байт даних, керуючи лінією SDAx, в той час як ведучий продовжує формувати такти. При цьому ведучий відпускає лінію SDAx, для того щоб нею міг керувати ведений.

Протягом зчитування, ведучий повинен закінчити прийом даних від веденого виконавши умову "NACK" на останньому байті повідомлення, який він бажає прийняти.

Ведучий посилає STOP, щоб закінчити повідомлення і повернути шину в неактивний стан.

Приклад використання протоколу I2C, який ініціалізує інтерфейс і налаштовує його на прийом 1 байту (або 3 байт) та запису прийнятих даних в змінну «data»:

```
Ініціалізація I2C1 на портах SCL: PB6 і SDA: PB7 з тактовою частотою 50kHz
    TM_I2C_Init(I2C1, TM_I2C_PinsPack_1, 50000);
    // Зчитати 3 байти даних з slave з адресою 0xD0
     //Перший регістр для зчитування за адресою 0x00
     //Зберегти отримані дані у змінній "data"
    uint8_t data[3];
    TM_I2C_ReadMulti(I2C1, 0xD0, 0x00, &data[0], 3);
    //записати мультибайти в  slave за адресою 0xD0
    //Записати до регістрів, починаючи з 0x00, отримати дані в змінній "data" і записати 3 байти
    uint8_t data[] = {0, 1, 2};
    TM_I2C_WriteMulti(I2C1, 0xD0, 0x00, &data[0], 3);
    //    Прикладдля I2C
    #include "stm32f4xx.h"
    #include "defines.h"
    #include "tm_stm32f4_i2c.h"
    #include "tm_stm32f4_delay.h"
    // Адреса Slave
    #define ADDRESS        0xD0 // 1101 000 0
    int main(void) {
       uint8_t data[] = {0, 1, 2};
       //Ініціалізаціясистеми
       SystemInit();
       //Ініціалізація I2C, SCL: PB6 і SDA: PB7 зтактовоючастотою 100kHz
```

```
TM_I2C_Init(I2C1, TM_I2C_PinsPack_1, 100000);
//Запис "5" заадресою0x00 для ведення з адресою ADDRESS
TM_I2C_Write(I2C1, ADDRESS, 0x00, 5);
// Записати мультибайти для ведення з адресою ADDRESS
// Записати до регістрів починаючи з  0x00, отримати дані зі знінної "data" і записати 3 байти
TM_I2C_WriteMulti(I2C1, ADDRESS, 0x00, data, 3);
//Прочитати 1 байт з  slave за адресою 0xD0 (1101 000 0) ірозташуваннямрегістра 0x00
data[0] = TM_I2C_Read(I2C1, ADDRESS, 0x00);
 //Зчитати 3 байти даних з slave за адресою 0xD0
 // Перший регістр, з якого зчитуємо знаходиться за адресою 0x00
 // Зберігажмо отримані дані у змінну "data"
TM_I2C_ReadMulti(I2C1, 0xD0, 0x00, data, 3);
while (1) {
}}
```

5. Опис інтерфейсу SPI

Послідовнийперіферійнийінтерфейс, шина SPI) - послідовнийсинхроннийстандартпередачіданихврежиміповногодуплексу, призначенийдлязабезпеченняпростогоінедорогогозв'язкумікроконтролеріві периферії. SPI такожінодіназиваютьчьотирьохпровіднимінтерфейсом:

MOSI – Передачаданихвід Master до Slave

MISO – Передача даних від Slave до Master

SCK – Через цей провід передається тактовий сигнал до Slave пристрою

NSS – Через цей провод Master дає зрозуміти Slave, що зараз буде відправка данних саме йому.

З опису всіх чотирьої ліній можна зробити наступні висновки

- SPI - це послідовний інтерфейс. Біти даних передаються один за одним
- SPI - це синхронний інтерфейс. Це означає, що передача даних (в будь-яку сторону) відбувається тільки в той час коли Master генерує імпульси синхронізації і передає їх через провід SCK інших пристроїв на шині.
- До однієї шини може підключатися кілька пристроїв, кількість яких теоретично не обмежена.
- SPI надає можливість обмінюватися даними в повнодуплексному режимі. Поки Master генерує тактові імпульси, він може посилати дані на Slave пристрій і одночасно приймати їх від нього ж.

Тепер розглянемо, як пристрої підключаються до шини. Стрілки показують напрям передачі сигналу:

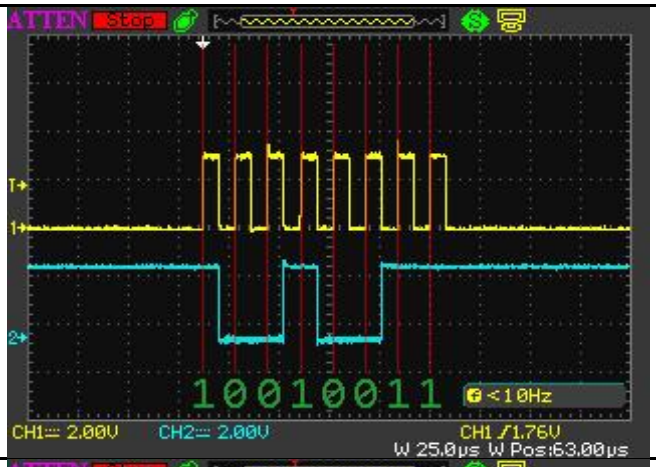
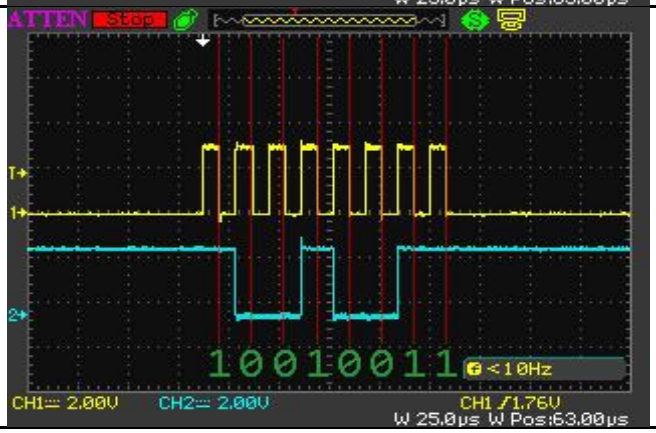
Рис. 8.4. Інтерфейс SPI

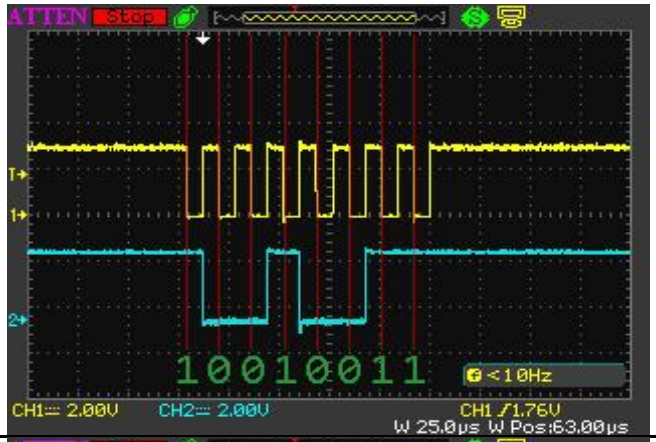
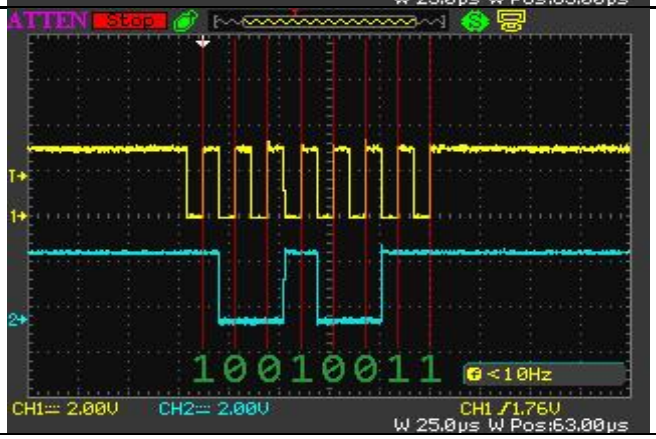
Як видно на рис 8.4, всі лінії інтерфейсу крім CS, об'єднуються між собою. Для кожного Slave пристрої, майстер має окремий вихід CS. Для того щоб обмінятися даними з іншим Slave пристроєм, Master встановить на ніжці CS2 низький логічний рівень, а на двох інших (CS1 і CS3) - високий. Таким чином Slave_1 і Slave_3 не передаватимуть інформацію, і тим самим не створять перешкод обміну інформацією Master і Slave_2. У такої схеми є один

недолік - на 10 Slave пристроїв, майстер повинен мати 10 окремих ніжок для підключення до CS.

Як уже згадувалося вище, до однієї шини можуть бути підключені самі різні slave пристрої, деякі з них досить швидкі і можуть обмінюватися даними з майстром на великій швидкості, а деякі - ні. Це означає, що Mater не повинен надто швидко генерувати тактові імпульси, інакше повільні Slave пристрої його не зрозуміють через спотворення даних. На таблиці 8.1 показані режими роботи SPI та їх відмінності один від одного. У всіх 4-х режимах, Master посилає один і той же байт (0x93). Жовта лінія це SCK, а синя - MOSI.

Таблиця 8.1

Режим	CPOL	CPHA	Осцилограма	Опис режиму
0	0	0		Вибірка по передьому наростаючому фронті
1	0	1		Вибірка по задньому спадаючому фронту

2	1	0		Вибірка по передньому спадаючому фронту
3	1	1		Вибірка по задньому наростаючом у фронту

Як видно з таблиці 8.1, номер режиму складається з двох біт - **CPOL** і **CPHA**. Біт **CPOL** визначає, в якому стані буде перебувати вивід SCL в той час коли нічого не передається. Якщо **CPOL** = 0, то в режимі простою на ніжці низький логічний рівень. Це означає, що передній фронтом буде вважатися перехід з 0 в 1. Якщо **CPOL** = 1, то в режимі простою на ніжці високий логічний рівень. Це означає, що передній фронтом буде вважатися перехід з 1 в 0 (а заднім фронтом, відповідно, навпаки з 0 в 1). Біт **CPHA** визначає по якому фронту потрібно робити вибірку 0 - по передньому фронту, 1 - по задньому фронту.

Наступний важливий параметр - порядок проходження біт. Зазвичай, першим передається старший біт, але іноді буває навпаки, якщо цього не врахувати, то можливі помилки при роботі інтерфейсу. Кількість біт може змінюватися, зазвичай це 8 біт. Далі розглянемо **SPI** на платі **STM32F4DISCOVERY**.

На рисунку 8.5 показані лінії SPI мікроконтроллера**STM32F407Vxxx**:

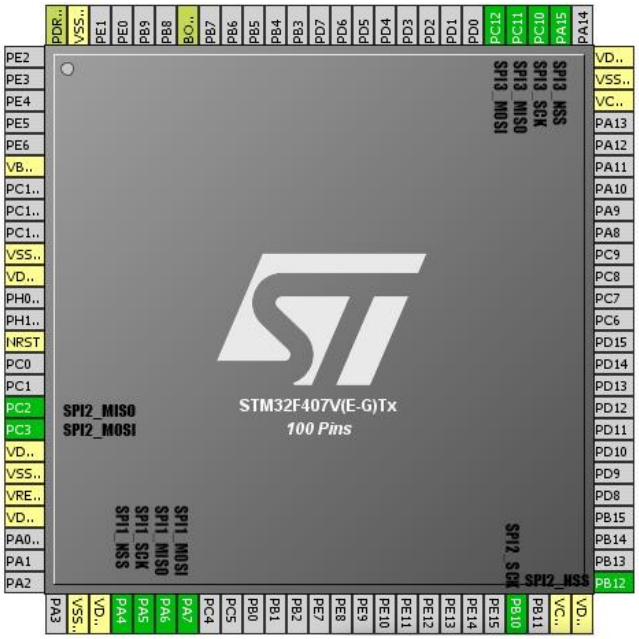


Рис. 8.5. Порти STM32F407Vxxx

Для спрощення роботи зі всією периферією мікроконтролера, (таймери, UART, SPI і т.д.) компанія ST розробила бібліотеку stdperiph_lib. З її використанням код стає більш зрозумілим і простішим для читання, також поліпшується перенесення коду з одного STM32 контролера на інший. Проте для кращого розуміння SPI розглянемо регістри:

Регістр SPI_CR1– регістр керування

Таблиця 8.2

BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIDIMODE – якщо цей біт встановлений, то прийом та передача даних здійснюються по одному проводу (не враховуючи SCK). При цьому, MOSI вивід Master підключається до MISO Slave.

BIDIOE – цей біт використовується в однопроводному режимі. Якщо він встановлений - SPI модуль тільки передає дані. Якщо скинутий - то приймає.

CRCEN – включає апаратний модуль розрахунку контрольної суми. 0 - викл, 1 - вкл. Змінювати стан цього біта можна тільки коли SPI модуль вимкнений (біт SPE = 0).

CRCNEXT – якщо цей біт встановлений, то після наступної передачі байту даних, буде відправлена контрольна сума.

DFF – якщо біт скинутий, то SPI модуль передає / приймає дані по 8 біт, інакше передається / приймається по 16 біт. Змінювати стан цього біта можна тільки коли SPI модуль вимкнений (біт SPE = 0).

RXONLY – якщо використовується 2-х провідний режим (див біт BIDIMODE) то установка цього біта забороняє передачу, SPI модуль працює тільки на прийом.

Перед описом наступних двох біт, потрібно відзначити одну цікаву особливість виведення NSS. Якщо SPI модуль налаштований в режимі Slave, то він може отримувати сигнал з вивода NSS або ж програмно. Якщо біт SSM скинутий, то сигнал SS буде зчитуватися з вивода NSS, а якщо він встановлений, то стан вивода NSS ігнорується. У такому випадку для керування сигналом SS переходить на біт SSI. Біт встановлений - є сигнал SS, інакше немає. Якщо ж SPI модуль працює в режимі майстра, то вивід NSS потрібно підтягнути до живлення або включити програмне управління (SSM = 1) і встановити біт SSI. В іншому випадку - SPI модуль вирішить, що з'явився новий Master і сам стане Slave.

LSBFIRST – задає порядок передачі біт: 0 - спочатку передається старший біт. 1 - спочатку передається молодший біт.

SPE – вмикає / вимикає SPI модуль

BR2,**BR1**,**BR0** – задають швидкість прийому / передачі (частоту SCK). Частота тактирования модуля SPI ділиться на число, яке задається комбінацією цих трьох біт.

Таблиця 8.3

BR2	BR1	BR0	Дільник
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

Змінювати стан цих біт можна тільки коли SPI модуль вимкнений (біт SPE = 0).

MSTR – якщо біт встановлений - SPI модуль є Master, інакше Slave.

CPOL – полярність сигналу SCK.
CPHA – фаза сигналу SCK.

Регістр SPI_CR2– регістр керування 2

Таблиця 8.4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	FRF	Res.	SSOE	TXDMAEN	RXDMAEN
								rw	rw	rw	rw		rw	rw	rw

TXEIE –дозволяє переривання, коли буфер передачі порожній.

RXNEIE – дозволяє переривання, коли буфер заповнений даними.

ERRIE – дозволяє переривання у разі виникнення помилки. Їх три, щоб розібратися яка виникла, потрібно дивитися стан біт в регістрі статусу.

SSOE – Якщо цей біт виставлений, то SPI модуль сам керує виводом NSS. Тобто перед початком передачі виставляє нуль на цьому виводі, а після завершення - виставляє одиницю.

TXDMAEN – дозволяє / забороняє запит DMA по завершенню передачі

RXDMAEN – дозволяє / забороняє запит DMA по завершенню прийому

Статусний регістр SPI_SR

Таблиця 8.5

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							FRE	BSY	OVR	MODF	CRC ERR	UDR	CHSID E	TXE	RXNE
							r	r	r	r	rc_w0	r	r	r	r

FRE – Frame error flag, він використовується, коли SPI модуль працює в режимі «TI mode» (біт FRF = 1).

BSY – якщо цей біт встановлений, то модуль SPI зараз зайнятий передачею даних.

OVR – біт виставляється в тому випадку, якщо в SPI модуль надійшли нові дані і замінили старі, що не були прочитані.

MODF – виставляється в тому випадку якщо Master раптово перестав бути Master. Таке можливо, коли вивід Master NSS налаштована як вхід і на неї надійшов сигнал низького рівня.

CRCERR – помилка контрольної суми

UDR – флаг не використовується в режимі SPI

TXE – Передача даних закінчилась

RXNE – Прийом даних завершено

Регістр SPI_DR– регістр данних

Представляє собою 16-ти бітний регістр даних, який розбитий на два. Один для передачі, а інший для прийому, але робота з ними здійснюється через один регістр SPI_DR. Якщо щось в нього записати, то запис даних проводиться в регістр для передачі. Якщо прочитати, то дані зчитуються з регістра для прийому даних.

Регістр SPI_CRCPR

У цей регістр записують деяке число, яке повинно вплинути на розрахунок контрольної суми. За замовчуванням записано число 7.

Регістр SPI_TXCRCR

У цей регістр записується контрольна сума, яка була розрахована для переданих даних.

Регістр SPI_RXCRCR

У цей регістр записується контрольна сума, яка була розрахована для прийнятих даних.

Приклад коду

Приклад використання SPI на платі **STM32F4DISCOVERY**, показує як налаштувати SPI1 в режимі Master і відправити дані. У нескінченному циклі відбувається передача одного і того ж байту (0x93).

```
01.#include "stm32f4xx.h"
02.#include "stm32f4xx_gpio.h"
03.#include "stm32f4xx_rcc.h"
04.#include "stm32f4xx_spi.h"
05.
06.int main(void) {
07.GPIO_InitTypeDef GPIO_InitStructure;
08.SPI_InitTypeDef SPI_InitStructure;
09.
```

```
10.// Тактуваннямодуля SPI1 іпортуA
11.RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
12.RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
13.
14.// Налаштовуємо вивід SPI1 для роботи в режимі альтернативної функції
15.GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);
16.GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
17.GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
18.
19.GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
20.GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
21.GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
22.GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
23.GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_6 | GPIO_Pin_5;
24.GPIO_Init(GPIOA, &GPIO_InitStructure);
25.
26.//Заповнюємо структуру с параметрами SPI модуля
27.SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex; //повнидуплекс
28.SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b; // передаємопо 8 біт
29.SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low; // Полярністьта
30.SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge; // фазатактовогосигналу
31.SPI_InitStructure.SPI_NSS = SPI_NSS_Soft; // Кереватистаномсигналу NSS програмно
32.SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32; // Передільник SCK
33.SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB; // Першимвідправляєтьсястаршийбіт
34.SPI_InitStructure.SPI_Mode = SPI_Mode_Master; // Режим - Master
35.SPI_Init(SPI1, &SPI_InitStructure); //Налаштовуємо SPI1
36.SPI_Cmd(SPI1, ENABLE); // Включаємомодуль SPI1....
37.
38.// Оскільки сигнал NSS контролюжться програмно, встановимо його в одиницю
39.// Якщо скинути його в нуль, то SPI модуль вирішить, що
40.// використовується мультимастерна топология і його перевели в Slave.
41.SPI_NSSInternalSoftwareConfig(SPI1, SPI_NSSInternalSoft_Set);
42.while(1) {
```

```
43.SPI_I2S_SendData(SPI1, 0x93); //Передаємбайт 0x93 через SPI1
44.while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET)
45.}}
```

6. Описінтерфейсу USART

UART – вузол обчислювальних пристроїв, призначений для організації зв'язку з іншими цифровими пристроями. Перетворює передані дані в послідовний вид так, щоб було можливо передати їх по цифровій лінії інших аналогічних пристроїв. Метод перетворення добре стандартизований і широко застосовувався в комп'ютерній техніці, оскільки часто виникає потреба під'єднати пристрій до комп'ютера для обміну даними з ним.Для обміну даними два пристрої повинні бути з'єднані так як показано на рисунку 8.6:

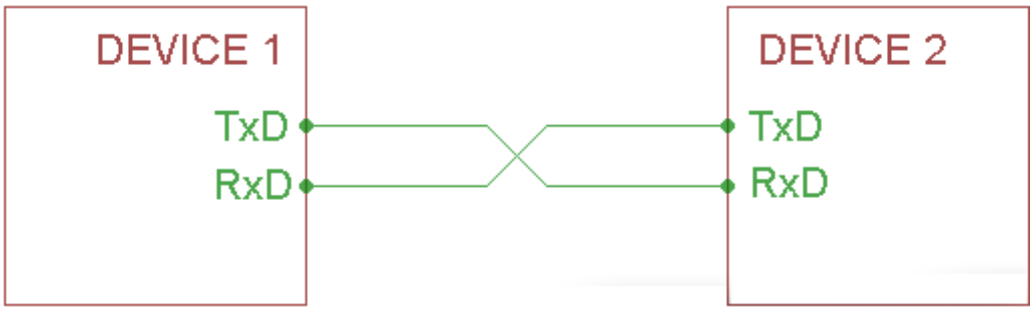


Рис.8.6. Зв’язок двох приладів через UART інтерфейс

Передавальнийвивід одного пристрою з'єднується з приймаючим виводом іншого, і навпаки. Обмін даними можливий одночасно в обидві сторони. Коли передача не здійснюється, на виході передавача завжди присутня логічна одиниця. Перед початком передачі даних передавач встановлює на виході логічний нуль. Це називається стартовим бітом, після якого починається передача біт даних, яких зазвичай вісім, але може бути від 5 до 9. Потім слід біт перевірки парності, якщо вона використовується. Цей біт призначений для запобігання обробки некоректних даних після прийому. Потім, після відправлення всіх біт йде стоповий біт, зазвичай один, можливі два. Стоповий біт завжди логічна одиниця, як показано на рисунку 8.7:

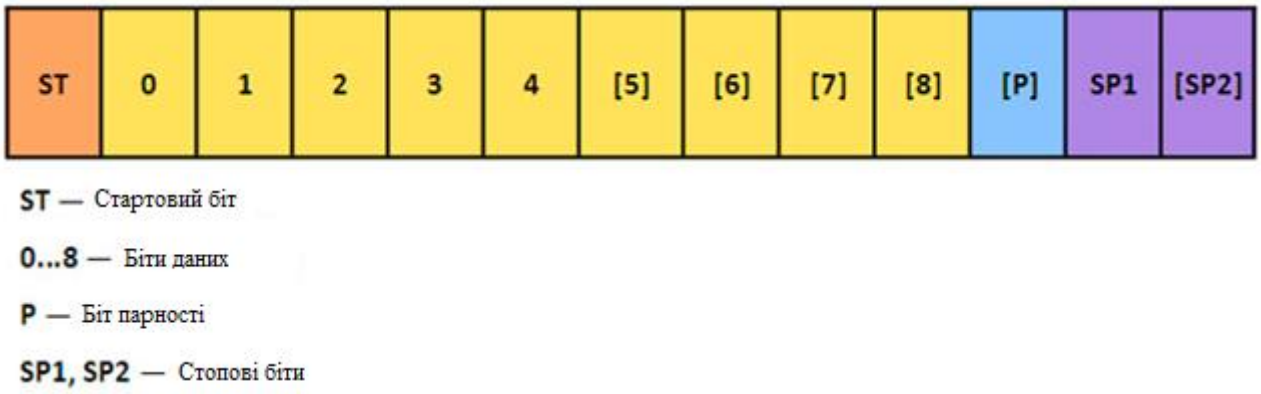
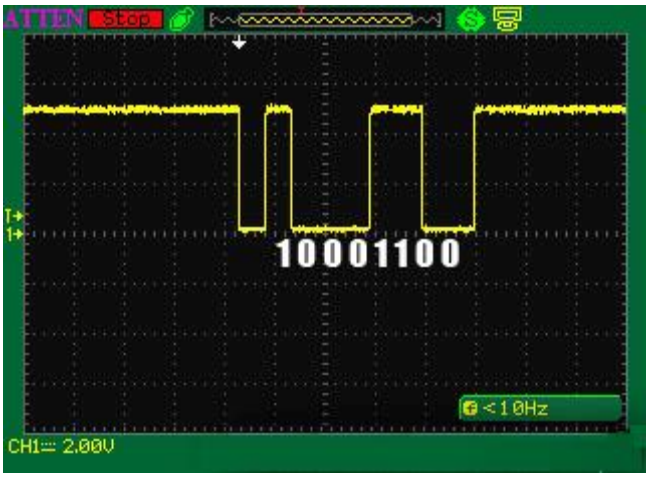


Рис. 8.7. Регістр передачі

Приклад передачі байту:



Рис, 8.8 Осцилограма передачі байту даних

Для успішної передачі даних, на обох пристроях, UART повинен бути налаштований з однаковими параметрами. Ще до початку передачі потрібно задати: Швидкість, кількість стопових біт, кількість біт даних, наявність перевірки парності. Швидкість передачі даних довільна, але як правило існує певний набір стандартних швидкостей: 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600 біт / сек і т.д. Чим нижче швидкість передачі даних - тим надійніше така передача.

У комп'ютера інтерфейсу UART як такого немає, але є COM порт. Різниця між COM портом і UART тільки в рівнях напруг. В COM порті логічний нуль - це приблизно +12 вольт, а логічна одиниця приблизно - 12 вольт. Якщо підключити мікроконтролер безпосередньо до виводів COM порту, то він може вийти з ладу. Для нього логічний нуль - це приблизно нуль вольт, а логічна одиниця - це приблизно повна напруга живлення. Для узгодження логічних рівнів використовують спеціальні мікросхеми, наприклад, max3232. Контролери STM32 працюють від 3.3 вольта, і напруга логічної одиниці не повинна перевищувати напруги живлення. Таким чином застосовуючи max232 потрібно знизити напругу виходу до 3.3 вольт. Або можна використовувати max3232. Схема включення показана на рисунку 8.9:

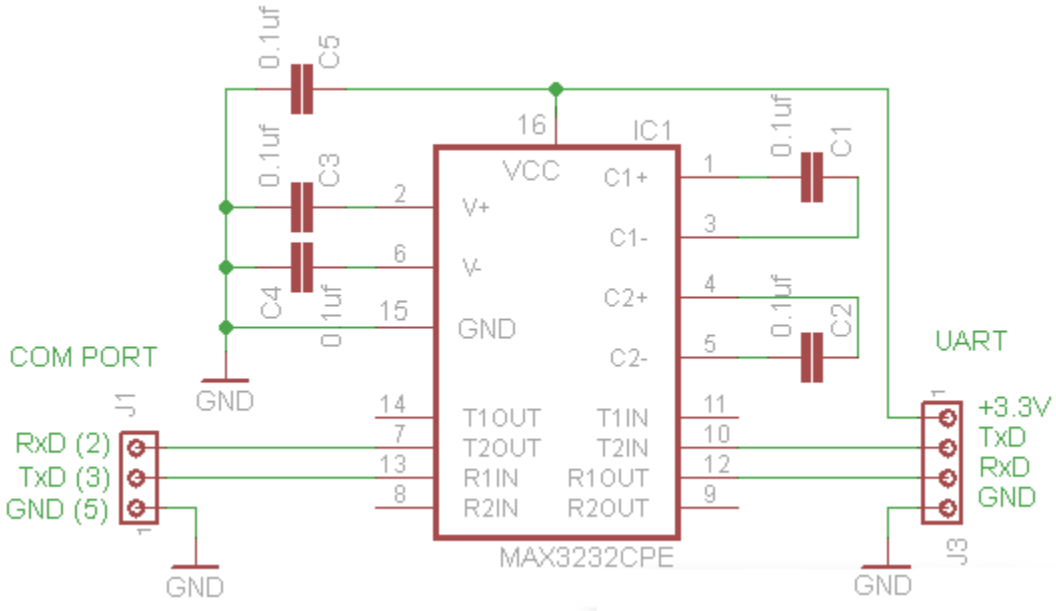


Рис. 8.9. Підключення мікросхемиmax3232CPE

Розглянемо програмну реалізацію. Для того щоб відправити щось в UART використовують термінальні програми. Наприклад – стандартний HyperTerminal, або Bray's Terminal(рисунок 8.10):

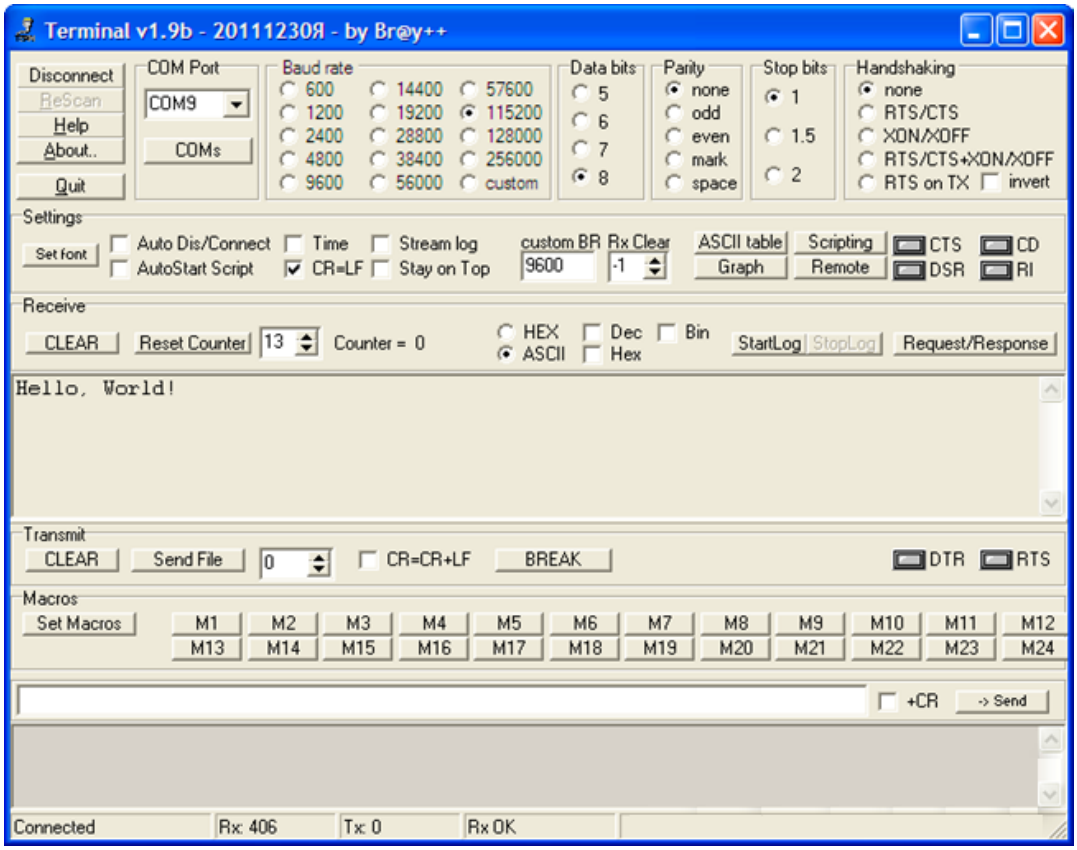


Рис.8.10. Вікно програми Bray`sTerminal

Регістри UART

USART_BRR– регістр швидкості прийому / передачі:

Таблиця 8.6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Регістр ділиться на дві частини: Ціла (**DIV_Mantissa**) і дробова (**DIV_Fraction**). Для отримання значення, яке потрібно записати в цей регістр, потрібно скористатись формулою:

USART_BRR = (fck + baudrate /2) / baudrate

де **fck** це частота тактування UART1, a baudrate це бажана швидкість передачі / прийому. В біти **DIV_Mantissa** слід записати ціле число (без округлення) отримане в результаті виконання арифметичної операції:

fck / (16*baudrate)

Дробову частину потрібно округлити до сотих і помножити на 16. Потім ще раз округлити, але вже до цілого. Після цього записати її в біти **DIV_Fraction**. **USART_CR1**– регістр керування

Таблиця 8.7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

UE - Біт призначений для включення інтерфейсу UART.

M - задає кількість біт даних, яке передаватиметься. Якщо біт дорівнює 0, тоді UART буде відправляти / приймати по 8 біт, якщо одиниці, то 9 біт.

PCE - Якщо біт дорівнює 1, то контроль парності включений, в іншому випадку вимкнений.

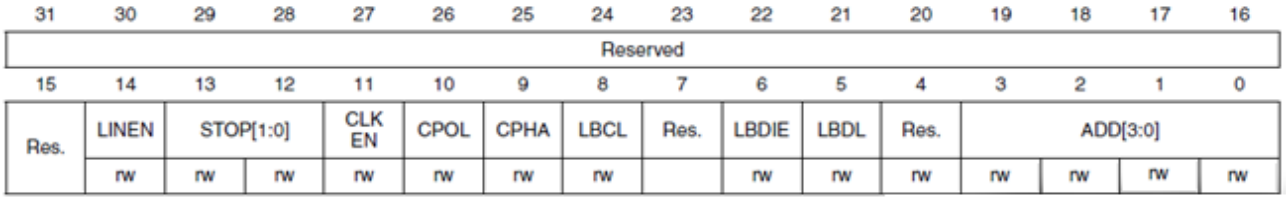
PS - Тип контролю парності: 0 - парне, 1 - непарне. **TE** - Якщо біт дорівнює 1, то дозволена робота передавача (вивід TxD)

RE - Якщо біт дорівнює 1, то дозволена робота приймача (вивід RxD)

Як видно по двох останнім бітам в цій таблиці - передавач і приймач один від одного не залежать. Якщо не потрібно приймати (або передавати) дані, то можна заощадити один вивід контролера (TxD або RxD) не включаючи не потрібну частину UART'а.

регістр налаштувань **USART_CR2**

Таблиця 8.8



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

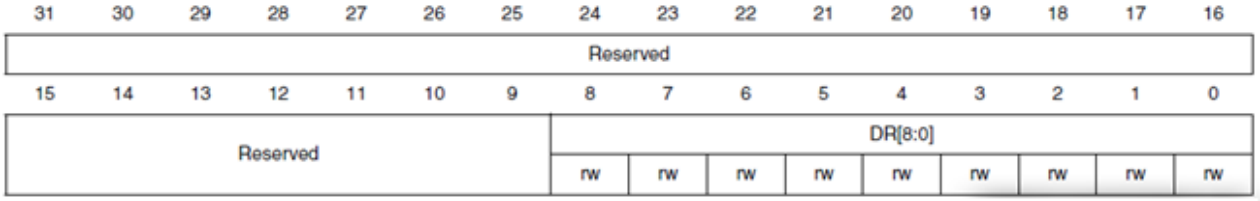
В цьому регістрі лише два біти які відносяться до UART: **STOP1**, **STOP0**

За допомогою цих двох біт можна задати кількість стопових біт в передачі.

STOP1	STOP0	Кількість стопових біт
0	0	1 стоп біт
0	1	0.5 стоп біта
1	0	2 стоп біта
1	1	1.5 стоп біта

В регістр даних, після того як всі налаштування UART зроблені можна щось відправити / отримати через UART. Для прийому / передачі служить регістр **USART_DR**:

Таблиця 8.9



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							DR[8:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

У ньому використовуються молодші 8 або 9 біт (залежно від біта **M** в регістрі **USART_CR1**). Щоб відправити у UART дані записуємо їх в цей регістр. Щоб прочитати дані читаємо цей регістр. При цьому, щоб не виникало ситуацій, зчитування при передачі, або запису при прийомі використовується регістр **USART_SR**– статусний регістр:

Таблиця 8.10

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

RXNE - Цей біт встановлюється, коли в UART щось прийшло. Якщо не використати з **USART_DR** дані, то вони перезапишуться новими.

TC - Якщо цей біт встановлений в одиницю, то це означає що передача даних завершена і можна записувати в регістр (**USART_DR**).

Приклад

Передача на ПК рядка з вітанням“Hello:)”. На вивід PA9 в даному прикладі був підключений вхід (RxD) USB-UART перетворювача. Налаштування UART: швидкість 9600 біт / сек, 1 стоп біт, без перевірки парності.

:

```
01.#include "stm32f10x.h"
02.#include "stm32f10x_gpio.h"
03.#include "stm32f10x_rcc.h"
04.
05.//Функція призначена для формування невеликої затримки
06.void Delay(void) {
07.volatile uint32_t i;
08.for (i=0; i != 0x70000; i++);
09.}
10.
11.//Функціящовідправляєбайтв UART
```

```
12.void send_to_uart(uint8_t data) {
13.while(!(USART1->SR & USART_SR_TC)); //Чекаємопокибіт TC врегістрі SR станет 1
14.USART1->DR=data; //Відправляємобайтчерез UART
15.}
16.
17.int main(void) {
18.GPIO_InitTypeDef PORTA_init_struct;
19.// Включаємо тактування порта A та USART1
20.RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1, ENABLE);
21.// Налаштовуємо вивід TxD (PA9) як вихід push-pull з альтернативною функцією
22.PORTA_init_struct.GPIO_Pin = GPIO_Pin_9;
23.PORTA_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
24.PORTA_init_struct.GPIO_Mode = GPIO_Mode_AF_PP;
25.GPIO_Init(GPIOA, &PORTA_init_struct);
26.//Налаштовуємо UART
27.USART1->BRR=0x9c4; //BaudRate 9600
28.USART1->CR1 |= USART_CR1_UE; //ДозволяємороботуUSART1
29.USART1->CR1 |= USART_CR1_TE; //Включаємопередавач
30.while(1) {
31.//Відправляємочерез UART слово Hello
32.send_to_uart('H');
33.send_to_uart('e');
34.send_to_uart('l');
35.send_to_uart('l');
36.send_to_uart('o');
37.send_to_uart(' ');
38.send_to_uart(':');
39.send_to_uart(')');
40.send_to_uart('\n');
41.Delay(); //невеликазатримка
```

42.}}

Створюємо порожній проект, копіюємо код, компілюємо і прошиваємо. Вікно терміналу, після виконання програми прийме вигляд як на рисунку 8.11:

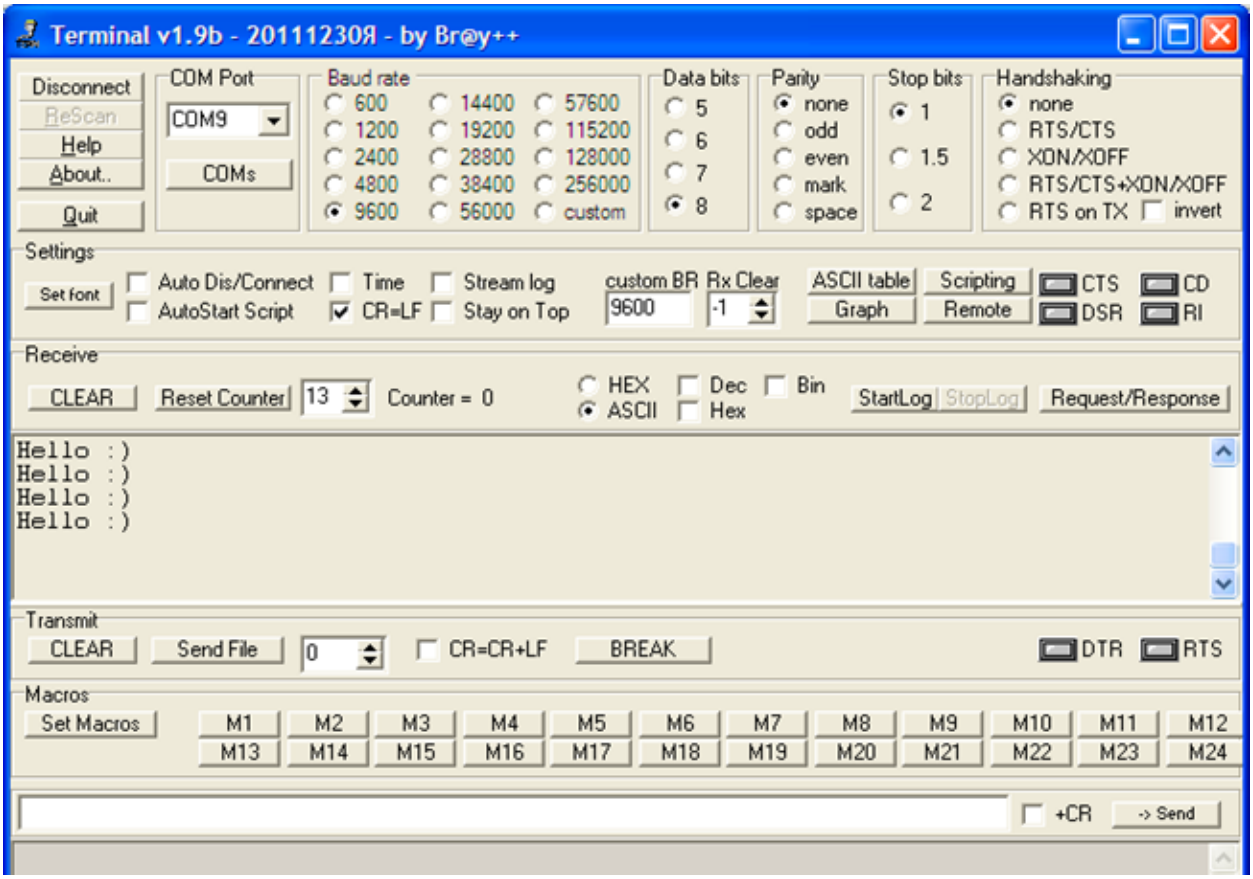


Рис.8.11. Прийом інформації з мікросхеми у вікні Bray`sTerminal

7. Опис інтерфейсу SDIO

SDIO - це інтерфейс для передачі даних в / з карт пам'яті. Розглянемо роботу з microSD флеш картою пам'яті по SDIO, який є в контролері **stm32f407vgt6** плати **stm32f4discovery**.

Працювати з картами флеш пам'яті можна за допомогою SPI інтерфейсу, але сучасні 32 бітні контролери, які мають модуль, спеціально призначений для роботи з картами пам'яті - SDIO, є значно дешевшими. Це також істотно спрощує і прискорює роботу. На малюнку представлені контакти і порівняння розмірів всіх SD карт:

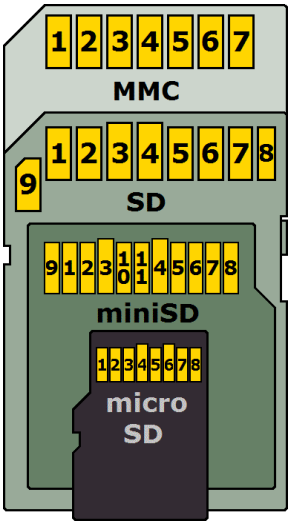


Рис.8.12 Контакти карти пам'яті

Призначення виводів:

Таблиця 8.11

MMC Pin	SD Pin	miniSD Pin	microSD Pin	Имя	I/O	Logic	Описание
1	1	1	2	NC	.	.	Не используется
2	2	2	3	CMD	I/O	PP, OD	Command, Response
3	3	3		VSS	S	S	Ground
4	4	4	4	VDD	S	S	Power
5	5	5	5	CLK	I	PP	Serial Clock
6	6	6	6	VSS	S	S	Ground
7	7	7	7	DAT0	I/O	PP	SD Serial Data 0
	8	8	8	NC nIRQ	. O	. OD	Не используется (memory cards) Interrupt (SDIO cards) (Negative Logic)
	9	9	1	NC	.	.	Не используется
		10		NC	.	.	Зарезервировано
		11		NC	.	.	Зарезервировано

З картою можна працювати по SDIO в режимі однобітної та 4-х бітної шини даних. Далі мова йтиме про одно бітну. Жодної принципової різниці, крім кількості використовуваних провідників, це не має. Як видно контактів на флешці досить багато. З них, крім живлення та землі, в основному потрібні три:

- CLK - тактування карти.

- CMD - по цій лінії передаються команди.
- DAT0 - лінія даних (у випадку 4х бітної їх буде 4).

З’єднуємо с контроллером:

- PC8 --- SDIO_D0 (DAT0)
- PC12 --- SDIO_CK (CLK)
- PD2 --- SDIO_CMD (CMD)

Передача даних з / на карту пов'язана з обчисленням контрольних сум. Це відбувається при кожній передачі. Розглянемо функцію ініціалізації модуля SDIO і ліній GPIO:

```
voidSD_LowLevel_Init(void) {
  uint32_t tempreg;

  GPIO_InitTypeDef GPIO_InitStructure;

  // Дозволяємо GPIOC та GPIOD Periph clock
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC | RCC_AHB1Periph_GPIOD, ENABLE);

  //Ініціалізуємопіни
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8, GPIO_AF_SDIO);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_SDIO);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource10, GPIO_AF_SDIO);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource11, GPIO_AF_SDIO);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource12, GPIO_AF_SDIO);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource2, GPIO_AF_SDIO);

  // Налаштовуємо PC.08, PC.09, PC.10, PC.11  піни: D0, D1, D2, D3
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 ; // | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11;
 //розкоментуватидля 4хбітноїшини
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz;
```

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOC, &GPIO_InitStructure);

// Налаштовуємо PD.02 CMD line
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_Init(GPIOD, &GPIO_InitStructure);

// Configure PC.12 pin: CLK pin
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOC, &GPIO_InitStructure);

//Enable the SDIO APB2 Clock
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SDIO, ENABLE);

// Enable the DMA2 Clock
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

//Ініціалізуємо SDIO зпочатковимтактуванням
HW_Flow Disabled, Rising Clock Edge, Disable CLK ByPass, Bus Width=0, Power save Disable
SDIO->CLKCR=tempreg;

// Включаємо SDIO
SDIO->>POWER = 0x03;
```

Функція передачі команди:

```
staticuint32_t SD_Command(uint32_t cmd, uint32_t resp, uint32_t arg) {
//Якаможебутивідповідь:
//0,2: Безвідповіді (cmdsent) ->NORESP
```

```
  //1:Короткавідповідь (cmdrend and ccrcfail) ->SHRESP
  //3:Довгавідповідь (cmdrend and ccrcfail) ->LNRESP

  //Очищуємофлаги
  SDIO->ICR=(SDIO_STA_CCRCFAIL | SDIO_STA_CTIMEOUT | SDIO_STA_CMDREND | SDIO_STA_CMDSENT);

  SDIO->ARG=arg;
SDIO->CMD=(uint32_t)(cmd & SDIO_CMD_CMDINDEX) | (resp & SDIO_CMD_WAITRESP) | (0x0400);

  //Блокуватипокинеотримаємовідповідь
  if(resp==NORESP) {
    //Чекаємона CMDSENT
    while(!(SDIO->STA & (SDIO_STA_CTIMEOUT | SDIO_STA_CMDSENT))) {};
  }
  else{//SHRESP or LNRESP or R3RESP
    //Чекаємона CMDREND чи CCRCFAIL
    while(!(SDIO->STA & (SDIO_STA_CTIMEOUT | SDIO_STA_CMDREND | SDIO_FLAG_CCRCFAIL))) {};
  }

  // Перевіряємо чи є відповідь правильною
  // Всі R3 відповіді без перевищення затримки вважаються правильними
  if(SDIO->STA & SDIO_STA_CTIMEOUT) {
    SD_Panic(cmd, "SDIO: Command Timeout Error\n");
  } elseif((SDIO->STA & SDIO_FLAG_CCRCFAIL) && (resp!=R3RESP)) {
    SD_Panic(cmd, "SDIO: Command CRC Error\n");
  }

  returnSDIO->STA;
}
```

8. Опис интерфейсу SAI

SAI - (Serial audio interface) інтерфейс передачі звукових сигналів. Складається з двох звукових підблоків, незалежних один від одного. Підтримує I2S, PCM / DSP і AC`97 протоколи. Може працювати в режимах Master / Slave. Звукові підблоки можуть як приймати, так і передавати, синхронно чи ні. Може використовуватися з DMA контролером, який може отримувати доступ до системної шини незалежно від центрального процесора.

Характеристики SAI

У таблиці наведені основні характеристики послідовного звукового інтерфейсу в **STM32F405xx**:

Таблиця 8.12

Symbol	Parameter	Conditions	Min	Max	Unit
f_{MCKL}	SAI Main clock output	-	256 x 8K	256xFs(2)	MHz
F_{SCK}	SAI clock frequency	Master data: 32 bits	-	64xFs	MHz
		Slave data: 32 bits	-	64xFs	
D_{SCK}	SAI clock frequency duty cycle	Slave receiver	30	70	%
$t_{v(FS)}$	FS valid time	Master mode	8	22	ns
$t_{su(FS)}$	FS setup time	Slave mode	2	-	
$t_{h(FS)}$	FS hold time	Master mode	8	-	
		Slave mode	0	-	
$t_{su(SD\_MR)}$	Data input setup time	Master receiver	5	-	
$t_{su(SD\_SR)}$		Slave receiver	3	-	
$t_{h(SD\_MR)}$	Data input hold time	Master receiver	0	-	
$t_{h(SD\_SR)}$		Slave receiver	0	-	
$t_{v(SD\_ST)}$ $t_{h(SD\_ST)}$	Data output valid time	Slave transmitter (after enable edge)	-	22	
$t_{v(SD\_MT)}$		Master transmitter (after enable edge)	-	20	
$t_{h(SD\_MT)}$	Data output hold time	Master transmitter (after enable edge)	8	-	

На рисунках рис.8.13 та рис.8.14 представлені часові діаграмми роботи SAIMaster і Slaveвідповідно.

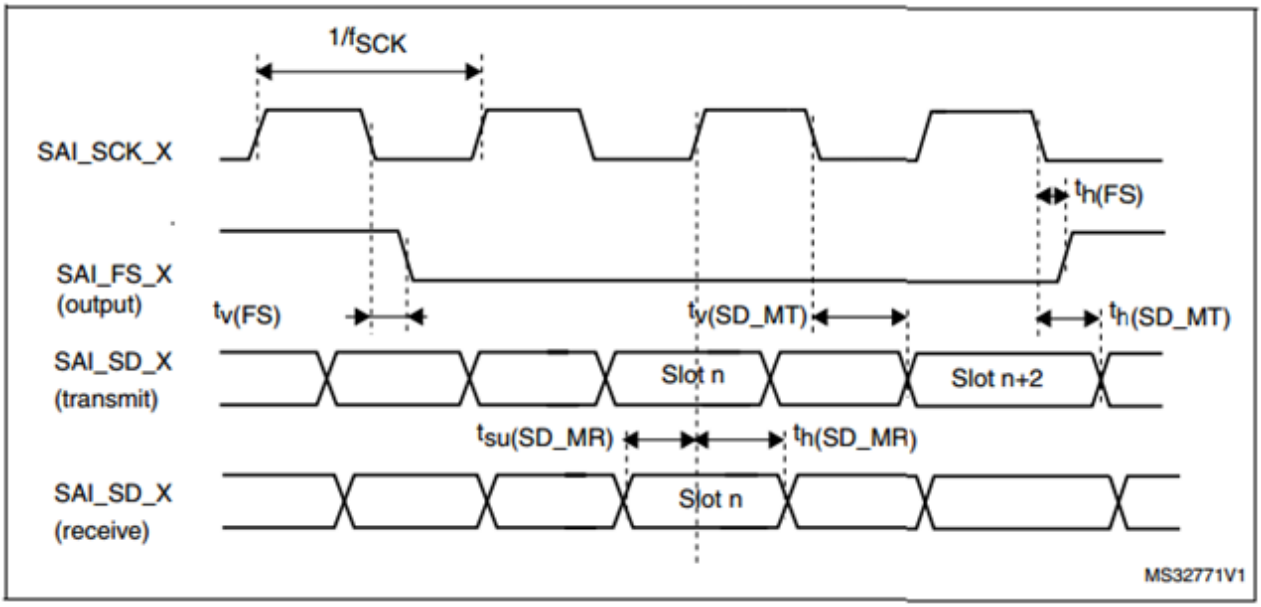


Рис.8.13 Часові діаграми SAIMaster

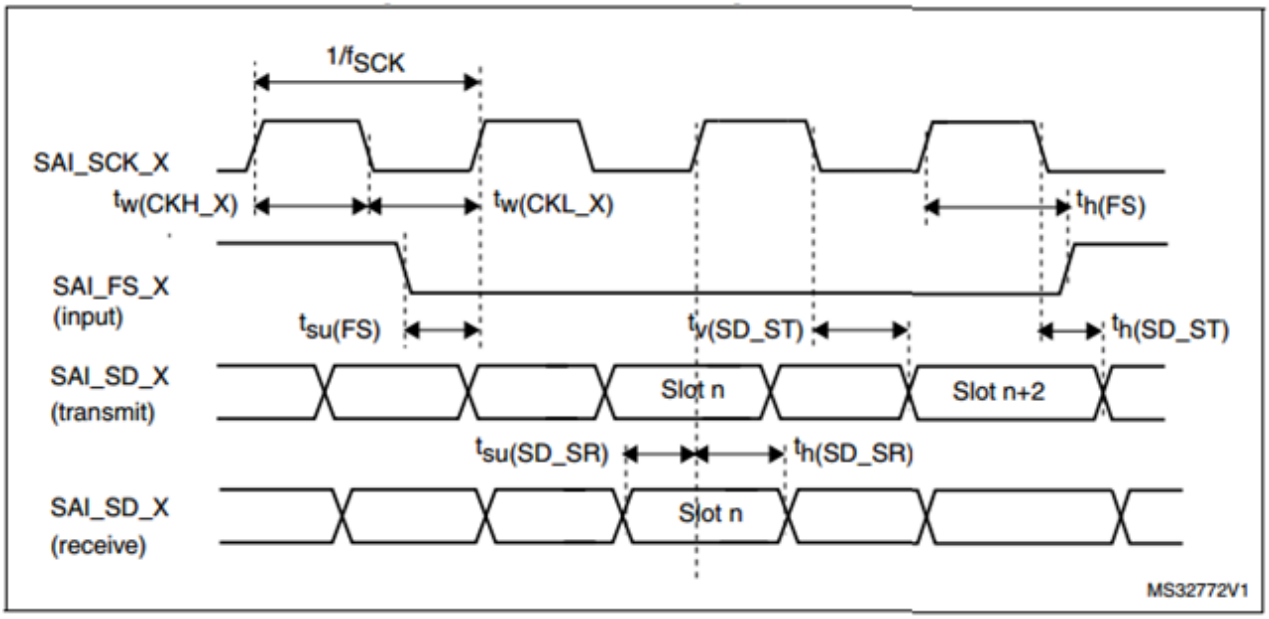


Рис.8.14. Часові діаграми SAISlave

Тема 3.9 Контролери Chrome-ART та FSMC контролери.

Сімейство високопродуктивних мікроконтролерів STM32F4 поповнилося новими представниками, що мають розширені графічні можливості. Нові контролери STM32F4x9, крім високопродуктивного ядра ARM Cortex-M4, працюючого на частоті до 180 МГц, володіють інтегрованим TFT-LCD-контролером, графічним прискорювачем Chrom- ART, підтримкою динамічної пам'яті SDRAM і послідовним аудіо-інтерфейсом SAI.

Існує величезна кількість областей, що вимагають графічного інтерфейсу з користувачем. У промисловості це - пульти операторів верстатів ЧПУ, пульти управління технологічними процесами та ін.; в системах «Розумний будинок» і системах безпеки – центральні модулі систем безпеки, панелі управління клімат-контролем (HVAC); в медицині - портативні медичні прилади (глюкометри, манометри, вимірювачі холестерину), системи штучної вентиляції легенів, комплексні системи моніторингу; в споживчій електроніці - панелі управління побутовою технікою, іграшки; у торгівлі - точки оплати, банкомати, сканери; в автомобільній електроніці - музичні системи, бортові комп'ютери, панелі приладів і т.д. Особливості нових контролерів представлені на рис.9.1.

Лінійка STM32F4x9 – це перші контролери сімейства STM32F4 з інтегрованим TFT-LCD-контролером і графічним прискорювачем Chrom-ART. Таке поєднання дозволяє значно знизити завантаження процесора і штатного DMA при обробці зображень. Варто відзначити, що зараз жоден з конкурентів STM32F4x9 не має апаратної підтримки такого широкого набору функцій (заливка прямокутної області, копіювання прямокутної області, копіювання з перетворенням формату кольору пікселів, змішування двох шарів і ін). Крім розширених графічних можливостей, особливостями STM32F4x9 є: високошвидкісний процесор ARM Cortex-M4, що працює на

частоті до 180 МГц, послідовний аудіо-інтерфейс Serial Audio Interface (SAI) і підтримка динамічної пам'яті SDRAM.

Найважливішим нововведенням лінійки є поява інтегрованого контролера TFT-LCD і графічного прискорювача Chrom-ART (DMA2D). Звичайно, контролери STM32 й раніше мали можливість управління TFT-LCD-дисплеями, але при цьому існували значні обмеження. Розглянемо більш детально переваги нового підходу до побудови графічних додатків на базі STM32F4x9.

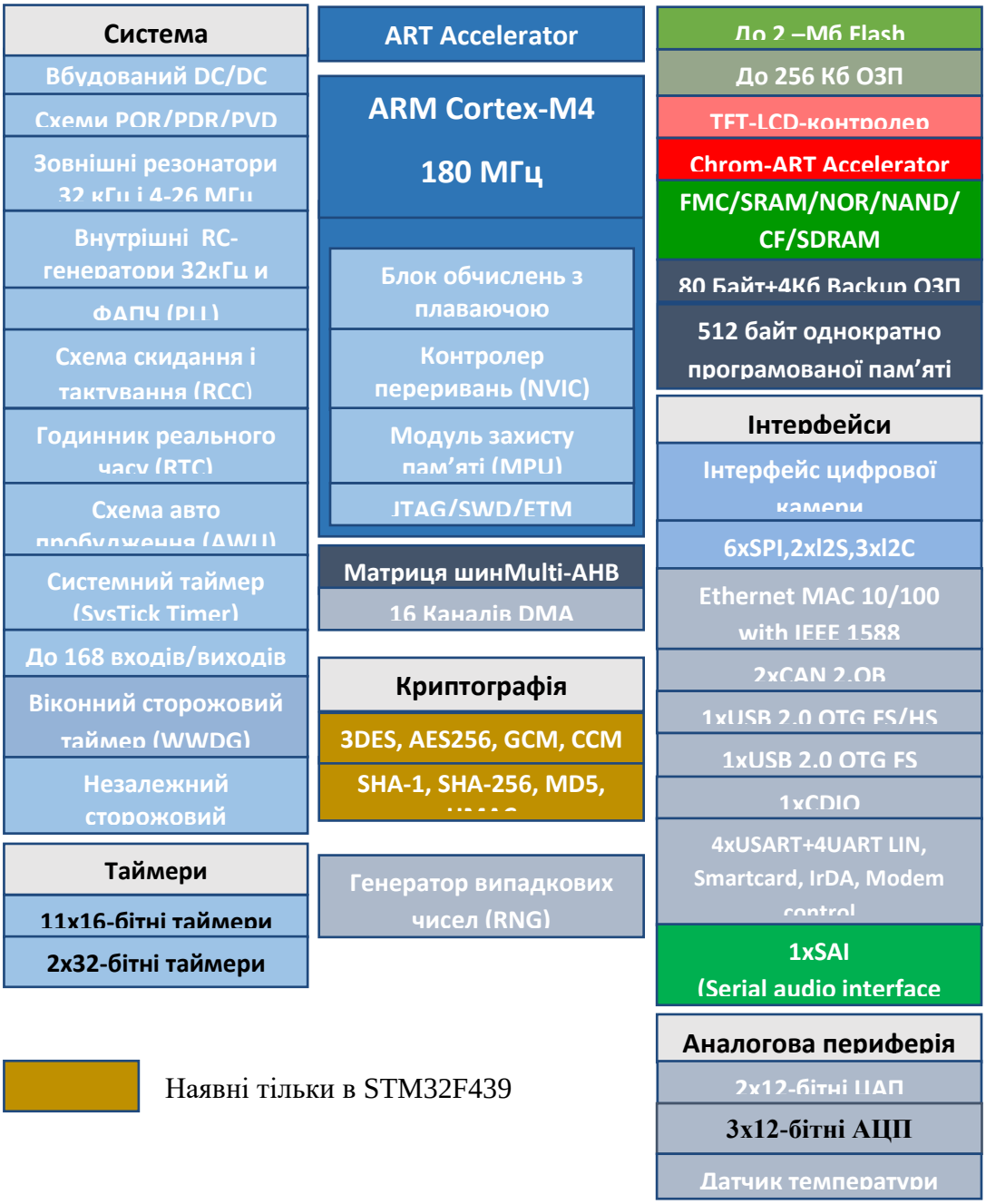


Рис.9.1. Особливості нових контролерів STM32F429 / STM32F439

1. Особливості графічних додатків на базі STM32.

Кольорові LCD-панелі вимагають спеціалізованого драйвера, який формує аналогові сигнали управління (рис.9.2). Управління самим драйвером може здійснюватися за допомогою різних інтерфейсів. Одним з основних і

широко поширеним засобом є використання RGB-інтерфейсу. До появи STM32F4x9 жоден контролер STM32 не мав апаратної підтримки RGB-інтерфейсу. Для взаємодії з TFT-панелями компанія STMicroelectronics пропонувала два варіанти.

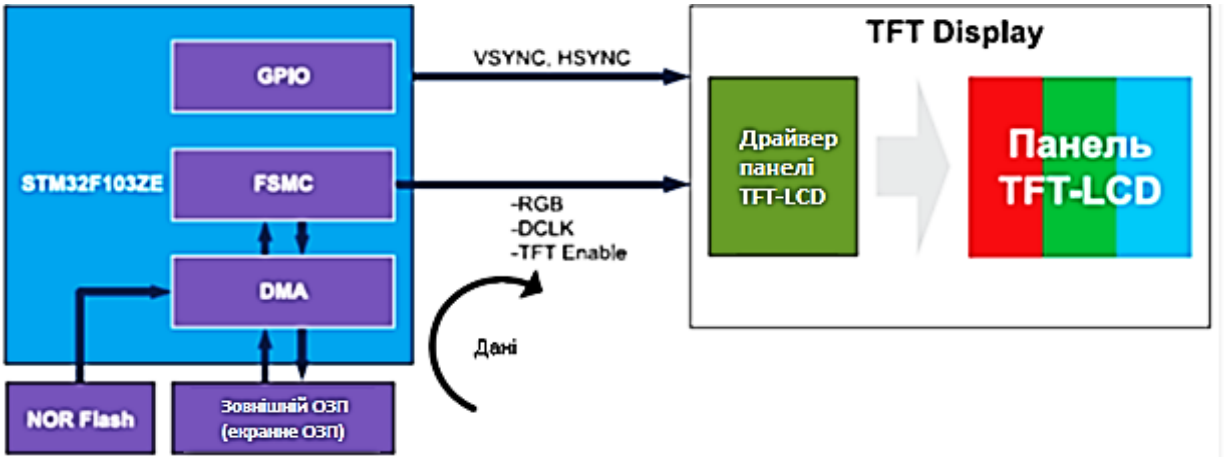


Рис.9.2. Програмна реалізація RGB-інтерфейсу

Варіант 1. Пряма реалізація RGB-інтерфейсу (рис.9.2). Контролер емулює роботу RGB-інтерфейсу за допомогою внутрішньої периферії. Даний варіант може бути ефективно реалізований для всіх контролерів STM32, що мають FSMC (Flexible Static Memory Controller).

Первісне зображення зберігається в зовнішній Flash-пам'яті. Для виведення на дисплей зображення за допомогою DMA переміщається під зовнішнє ОЗП, який виконує функції екранного шару (або шарів). Зображення на дисплеї оновлюється безпосередньо з цього ОЗП. Сигнали синхронізації VSYNC і HSYNC реалізуються за допомогою виходів загального призначення. В якості шин даних (R, G, B) використовуються сигнали FSMC [D0: D15]. для сигналу DCLK зручно використовувати FSMC WE в інверсному режимі. Очевидні й переваги, і недоліки даної реалізації. Переваги: низька вартість реалізації і мала зайнятість процесора при виводі статичних зображень. Використання апаратної периферії (DMA, FSMC) дозволяє знизити завантаження процесора при статичній картинці на екрані до 1%. Недоліки:

неможливість отримання складних або динамічних зображень. Дійсно, так як відсутні апаратні засоби обробки зображень, то процесор витрачає все обчислювальні потужності на графіку.

Варіант 2. Використання зовнішнього контролера TFT-LCD. Контролер TFT-дисплея виконує ряд функцій: обмінюється даними з мікроконтролером (по SPI, Intel 8080, Motorola 6800); зберігає отримані дані, виконуючи функції екранного ОЗП; управляє драйвером TFT-панелі по RGB-інтерфейсу; виконує деякі функції обробки зображень (поворот / дзеркальне відображення, визначення активної області та ін.). Даний варіант може бути ефективно реалізований для всіх контролерів STM32, що мають FSMC (Flexible Static Memory Controller). Для дисплеїв з невеликою роздільною здатністю зображення іноді буває достатньо SPI-інтерфейсу. Вихідне зображення зберігається у зовнішній Flash пам'яті (рис.9.3).

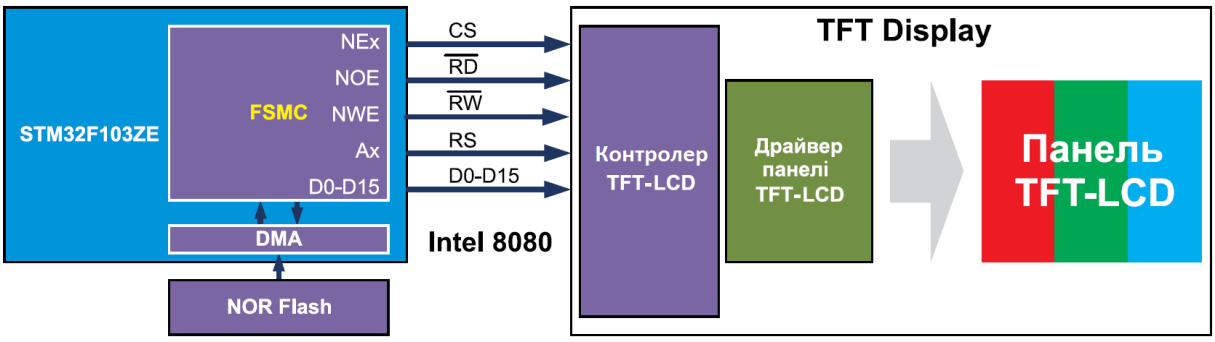


Рис.9.3. Використання зовнішнього TFT-LCD-контроллера

Зображення з Flash пам'яті за допомогою FSMC- контролера передається в TFT-LCD-контролер. При цьому останній виконує функції екранного ОЗП, тому немає необхідності в зовнішньому ОЗП, як в попередньому варіанті. Серед переваг використання зовнішнього графічного контролера можна відзначити: низьку вартість кінцевого пристрою, відсутність зовнішнього ОЗП, простоту реалізації друкованої плати.Недоліки такі ж, як і у варіанті з прямою реалізацією RGB-інтерфейсу: мала придатність для складних і динамічних зображень. Таким чином, до появи STM32F4x9

можна було створювати нескладні графічні додатки, а головним недоліком була велике завантаження процесора при їх обробці. В STM32F4x9 ця проблема була вирішена додаванням інтегрованого багатофункціонального TFT-LCD- контролера і графічного прискорювача Chrom-ART. Нові мікроконтролери STM32F4x9 володіють ефективними інструментами обробки зображень: TFT-LCD-контролером, графічним прискорювачем Chrom-ART, контролером зовнішньої пам'яті FMC (Flexible external memory controller). Взаємодія такого контролера з TFT дисплеєм зображена на рис.9.4.

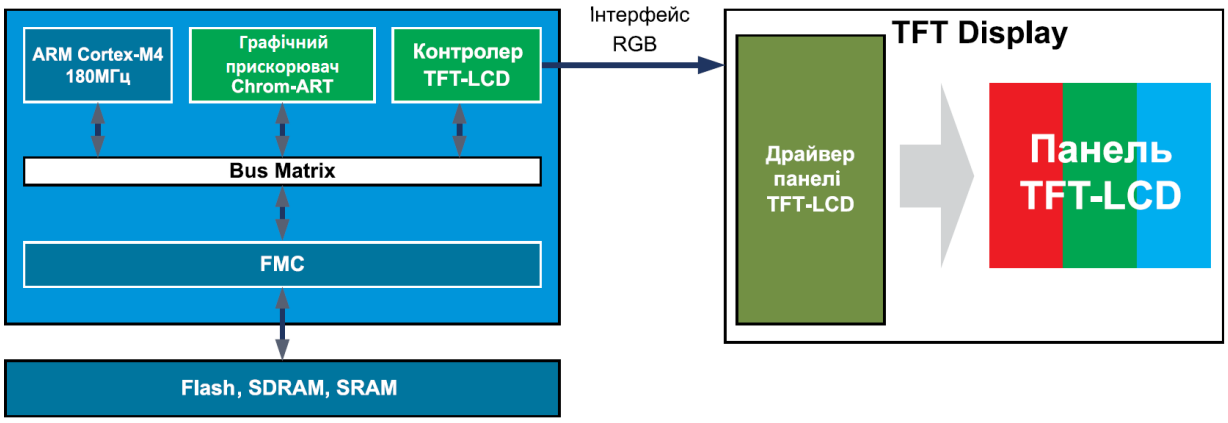


Рис. 9.4. Взаємодія STM32F4x9 з TFT-дисплеєм

Графічний прискорювач Chrom- ART (DMA2D) являє собою спеціалізований DMA, створений для роботи з графічними зображеннями. DMA2D є майстром на шині AHB і здатний працювати з різними форматами даних (8-/16-/32-бітними).

DMA2D виконує ряд операцій:

• заливка вихідного зображення (або його частини) заданим кольором;

• копіювання вихідного зображення (або його частини) у задану область кінцевого зображення;

• копіювання вихідного зображення (або його частини) у задану область кінцевого зображення з додатковим перетворенням формату кольору пікселів.

• змішування зображень (або їх частин) з однаковими або різними форматами кольорів пікселів і розташування в задану область кінцевого зображення з додатковим перетворенням формату кольору пікселів.

2. Контролер FMC

Додаткову гнучкість при побудові графічних додатків дає використання зовнішньої пам'яті різних типів. Новий контролер зовнішньої пам'яті FMC здатний підтримувати не тільки статичну пам'ять (NOR / PSRAM, NAND / PC Card memory), але і динамічну (SDRAM).

Основні особливості DMA2D:

•Проста AHB master-архітектура.

•Програмований AHB slave-інтерфейс з підтримкою 8/16/32-бітних звернень (за винятком CLUT звернень, які є 32-бітними).

• Програмований користувачем розмір робочої області

• Програмовані користувачем зміщення джерел і призначення зон

• Програмовані користувачем джерела і адреси призначення на всій пам'яті

• До 2-х джерел з операції змішування

• Альфа-значення може бути змінено (початкове значення, фіксоване значення або модулююче значення)

• Програмовані користувачем вихідний і кінцевий формат кольору

• Підтримувані формати до 11 кольорів -від 4 біт до 32 біт на піксель з прямим і непрямимкодуванням кольору

• 2 внутрішніх слотів пам’яті для зберігання CLUT (Color Look-Up Table) в режимі непрямих кольорів

• Автоматичне CLUT завантаження або CLUT програмування через ЦП

• Програмовані користувачем CLUT розміри

• Вбудований таймер для контролю пропускної здатності AHB

• 4 режими роботи: регістр-пам'ять, пам'ять-пам'ять, пам'ять-пам'ять зпіксельним форматом перетворення та пам'ят-пам'ят з піксельним форматом перетворення ізмішуванням

• Заповнення області фіксованим кольором

• Копіювання з однієї області в іншу

• Копіювання з піксельного формату між джерелом і приймачем зображення

• Копіювання від двох джерел з незалежним форматом кольору і змішування

• Переривання та зупинення DMA2D операцій

• Генерація переривань на шині помилок або конфлікту доступу

• Генерація переривань по завершенні процесу

• Водяний знак переривання користувача програмований лінією призначення

Цифровий водяний знак (ЦВЗ)- технологія, створена для захисту авторських прав мультимедійних файлів. Зазвичай цифрові водяні знаки невидимі. Однак ЦВЗ можуть бути видимими на зображенні або відео. Зазвичай це інформація являє собою текст або логотип, який ідентифікує автора.

Невидимі ЦВЗ впроваджуються в цифрові дані, але не можуть бути сприйняті як такі. Найважливіше застосування цифрові водяні знаки знайшли в системах захисту від копіювання, які прагнуть запобігти або утримати від несанкційованого копіювання цифрових даних.Стеганографія застосовує ЦВЗ, коли сторони обмінюються секретними повідомленнями, впровадженими в цифровий сигнал.

3. Функціональний опис DMA2D

Контролер DMA2D виконує пряму передачу пам'яті. В якості ведучого AHB, може мати контроль над AHB шиною матриці ініціюючи AHB операції.

Контролер DMA2D

Контролер DMA2D налаштовується за допомогою регістру управління DMA2D (DMA2D_CR) який дозволяє вибрати:

• Вибирпти режим роботи

• Включення / відключення DMA2D переривання

• Старт / призупинення / припинення триваючої передачі даних

Slave-порт AHB використовується для програмування контролера DMA2D. Структурна схема DMA2D показана на рис.9.5.

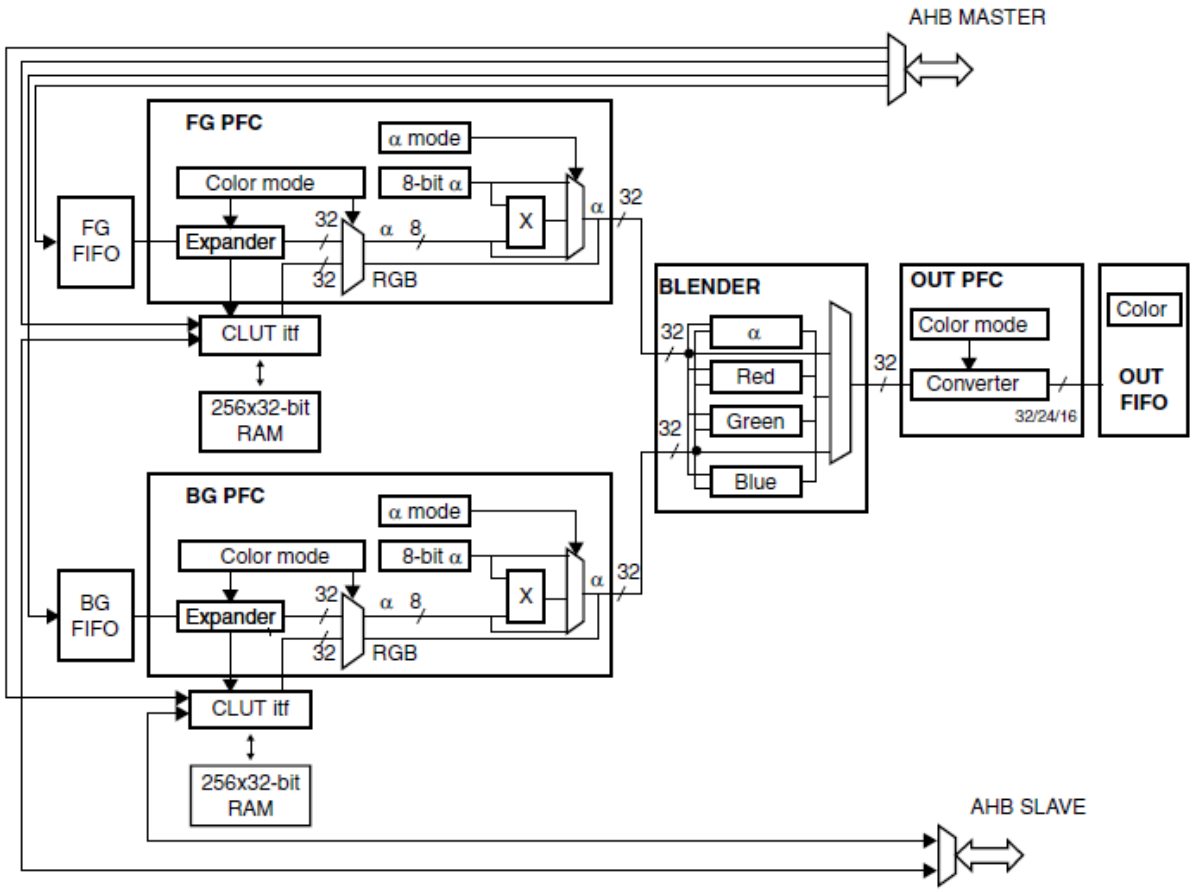


Рис.9.5. Структурна схема DMA2D.

4. Стекі FIFOFGта FIFOBG

Схема DMA2Dмає два FIFO: в активному режимі (FG) і в фоновому режимі (BG), якіотримують вхідні дані для подальшого відтворення та/або обробки.

У стекінадходять пікселі відповідно до формату кольору, визначеного у відповідних пікселяхформатуконвертерів: FGPFC та BGPFC.

Вони програмуються за допомогою набору регістрів управління:

• DMA2D_FGMAR регістр адреси пам'яті активного режиму DMA2D

• DMA2D_FGOR регістр зсуву активного режиму DMA2D

• DMA2D_BGMAR регістра пам'яті фонового режиму DMA2D

• DMA2D_BGBORрегістр зсуву фонового режиму DMA2D

• DMA2D_NLRрегістра кількості рядків і кількості пікселів на лініїDMA2D.

Коли DMA2D працює в режимі регістр-пам'ять, жоден з FIFO не перебува в активному режимі.Коли DMA2D працює в режимі пам'ять-пам'ять (ні без перетворення формату пікселів, ні без операції змішування), тільки FG FIFO активується і діє як буфер.

Коли DMA2D працює в режимі пам'ять-пам'ять з перетворенням формату пікселів (без операції змішування), BG FIFO не активовано.

Конвертори форматупікселів активного (FGPFC) та фонового (BGPFC) режимів

Конвертери формату пікселів активного режиму (FG PFC) і фонового режиму (BG PFC) виконують конвертування формату пікселя для генерації 32-бітного значення напіксель. PFC може такожзмінити альфа-канал.

Альфа-композиція позначає процес комбінування зображення з фоном з метою створення ефекту часткової прозорості. Цей метод часто застосовується для багатопрохідної обробки зображення по частинах з наступною комбінацією цих частин в єдине двовимірне результуюче зображення. Таким чином, альфа канал являє собою порожній простір, або просто прозорість.

Розрахунок яскравості результуючого пікселя після накладення двох пікселів один на одного виконується за формулою:

Result = Background * (1 - Alpha) + Foreground * Alpha

або

Result = Background + (Foreground - Background) * Alpha, де

Background - яскравість фонового пікселя α_{BG},

Foreground - яскравість накладаючогося пікселя α_{FG},

Alpha ∈ [0..1] - прозорість накладаючогося пікселя.

На першому етапіконвертор перетворює формат кольору. Оригінальний формат кольору пікселів активного режиму та фонового, налаштовуються в CM[3:0] бітах DMA2D_FGPFCCR і DMA2D_BGPFCCR, відповідно, наступній таблиці 9.1.

Таблиця 9.1

CM[3:0]	Color mode
0000	ARGB8888
0001	RGB888
0010	RGB565
0011	ARGB1555
0100	ARGB4444
0101	L8
0110	AL44
0111	AL88
1000	L4
1001	A8
1010	A4

Формат кольору кодуються наступним чином:

• Значення поля альфа-канала: прозорість

Значення 0xFF відповідає непрозорому пікселю, а 0x00 - прозорому.

• R поле для червоного кольору

• G поле для зеленого кольору

• B поле для синього кольору

• L поле: яскравість

Це поле індекс до CLUT для отримання трьох/чотирьох компонентів RGB/ARGB.

5. FGiBG CLUT інтерфейси

Інтерфейс CLUT управляє доступом до пам'яті і автоматичним завантаженням таблиці CLUT.

Можливітри види доступу:

• CLUT зчитуєтьсяPFC під час виконання операції перетворення формату пікселів,

• CLUT доступна через AHB-slave порт, коли процесор зчитує або записує дані в CLUT

• CLUT записується через основний AHB порт, коли виконане автоматичне завантаження CLUT.

Формат CLUT може бути 24 або 32 бітним. Він налаштовуєтьсяза допомогою CCM – біт регістра DMA2D_FGPFCCR (FG CLUT) або регістра DMA2D_BGPFCCR (BG CLUT), як показано в таблиці 9.2:

Таблиця 9.2

CCM	CLUT color mode
0	32-bit ARGB8888
1	24-bit RGB888

DMA2D BLENDER

DMA2D змішувач поєднує вихідні пікселі по парі з активного режиму (FG) та фонового режиму (BG) для обчислення результуючого пікселя.Змішування проводять у відповідності з наступним рівнянням:

$$\alpha_{OUT} = \alpha_{FG} + \alpha_{BG} - \alpha_{Mult}, \quad (1)$$

Де

$$\alpha_{Mult} = \frac{\alpha_{FG} \cdot \alpha_{BG}}{255} \quad (2)$$

$$C_{OUT} = \frac{C_{FG}.\alpha_{FG} + C_{BG}.\alpha_{BG} - C_{BG}.\alpha_{Mult}}{\alpha_{OUT}},$$

де C=R абоG абоB

Змішувачу не потрібна конфігурація регістра. Використання змішувача залежить від визначення режиму DMA2D в режимі MODE [1:0] поля регістру DMA2D_CR.

OUT PFC

Вихід OUT PFC виконує перетворення формату пікселів з 32 бітного до вихідного форматувизначеного в CM [2:0] області конфігурації перетворювача вихідного формату пікселя DMA2D регістру (DMA2D_OPFCCR).Формати виведення наведені в таблиці 9.3:

Таблиця 9.3

CM[2:0]	Color mode
000	ARGB8888
001	RGB888
010	RGB565
011	ARGB1555
100	ARGB4444

OUT FIFO

Вихід OUT FIFO програм – пікселі відповідає за формат кольору, визначеного на виході

PFC.

Область призначення визначена за допомогою набору регістрів управління:

• DMA2D адреса вихідного регістра пам'яті (DMA2D_OMAR)

• DMA2D вихід регістра зсуву (DMA2D_OOR)

• DMA2D кількість рядків регістру (кількість рядків і кількість пікселів на лінії) (DMA2D_NLR)

Якщо DMA2D працює в режимі регістр-пам'ять, налаштований вихідний прямокутник заповнюєтьсяза кольором, вказаним у вихідному регістрі

DMA2D кольором (DMA2D_OCOLR), який міститьфіксоване 32-бітне, 24-бітне або 16-бітне значення. Формат вибирається за допомогоюCM[2:0] областіDMA2D_OPFCCR регістра.

AHB -master

Вголовний AHB порт вбудований8-розрядний таймер, щоб забезпечити можливість обмеженняпропускної здатності на планці.

Цей таймер тактується AHB годинником і підраховує часу спокою між двома послідовнимидоступами. Це обмежує використання смуги пропускання.

Таймер включення і значення часу спокою настроюється за допомогою головного AHB порта через регістр конфігурації часу (DMA2D_AMPTCR).

DMA2D транзакції

DMA2D транзакції складаються з послідовності заданих числа передачі даних. Кількість даних і ширина можуть бути запрограмовані за допомогою програмного забезпечення.Кожна передача даних DMA2D складається з 4 етапів:

1. Завантаження даних з комірки пам'яті, на який вказує регістр DMA2D_FGMAR і

перетворення формату пікселя, як визначено в DMA2D_FGCR.

2. Завантаження даних з області пам'яті, на який вказує регістр DMA2D_BGMAR і

перетворення формату пікселя, як визначено в DMA2D_BGCR.

3. Змішування всіх знайдених точок відповідно до результатів альфа-каналівідповідно до операцій PFC з альфа-значеннями.

4. Перетворення формату результуючого пікселя за даними реєстру DMA2D_OCRта програмування даних в комірку пам'яті адресовану через регістр DMA2D_OMAR.

Конфігурація DMA2D

Вхідні і віхідні даніможуть призначатися периферійним пристроям і пам'яті на всю область пам'яті в 4 Гбайт, за адресами в діапазоні від 0x0000 0000 до0xFFFF FFFF. DMA2D може працювати в будь-якому з чотирьох наступних режимах обраних через MODE [1:0] біти регістра DMA2D_CR:

• Регістр-пам'ять

• Пам'ять-пам'ять

• Пам'ять-пам'ять з PFC

• Пам'ять-пам'ять з PFC і змішування

Регістр-пам'ять

Режим регістр-пам'ять використовується для заповнення користувачем певної області із заданим кольором.

Формат кольору встановлюється в регістрі DMA2D_OPFCCR.

DMA2D не виконує будь-які вибіркиданих з будь-якого джерела. Контролер DMA2D записує значення кольорувизначеного в регістрі DMA2D_OCOLR в область розташовану за адресою, яка позначаєтьсяDMA2D_OMAR і визначається в DMA2D_NLR і DMA2D_OOR.

Пам'ять-пам'ять

В режимі пам'ять-пам'ять, DMA2D не виконує ніяких перетворень графічних даних. FGFIFO діють як буфер, і дані передаються від комірки пам'яті джерела, визначеної в DMA2D_FGMAR до пам'яті призначення, розташування якої вказано в DMA2D_OMAR. Режим кольору задається в CM [3:0] бітами регістру DMA2D_FGPFCCR і визначає число бітів на піксель для обох входів і виходів. Розмір області передачі визначається регістрами DMA2D_NLR і DMA2D_FGOR для входу, і регістрами DMA2D_NLR і DMA2D_OOR длявиходу.

Пам'ять-пам'ять з PFC

В цьому режимі, DMA2D виконує перетворення формату пікселя з вихідних даних і зберігаєїх в комірці пам'яті призначення.Розмір області, що

підлягає передачі визначається DMA2D_NLR і DMA2D_FGORрегістрами для джерела, і DMA2D_NLR і DMA2D_OOR регістрами дляпризначення.Дані вибираються з місця, визначеного в регістрі DMA2D_FGMAR і оброблюються по сценарію PFC. Оригінальний формат пікселів налаштовується за допомогоюDMA2D_FGPFCCR регістра. Якщо вихідний формат пікселів знаходиться в режимі прямого кольору, то всі каналикольорів розширенідо 8 біт.Якщо формат пік селів – внепрямому кольоровому режимі, то пов'язана CLUT повинен бути завантажена вCLUT пам'яті.

Завантаження таблиці CLUT може бути виконано автоматично за наступним алгоритмом:

1. Встановити адресу CLUT в DMA2D_FGCMAR.

2. Встановити розмір CLUT в CS [7:0] бітах регістра DMA2D_FGPFCCR.

3. Встановити формат CLUT (24 або 32 біт) в CMM бітах регістра DMA2D_FGPFCCR.

4. Запустити CLUT завантаження, встановивши час початку регістра DMA2D_FGPFCCR.

Після того, як завантаження CLUT буде завершено, CTCIF прапор в регістрі DMA2D_IFR встановлено і якщо біт CTCIE встановлений в DMA2D_CR, то генерується переривання. Автоматичнийпроцес завантаження CLUTне може працювати паралельно з класичною передачою DMA2D. CLUT також можуть бути заповнені процесором або будь-яким інший хостом через порт APB. Доступ до CLUT неможливий, коли триває передача DMA2D, і використовує CLUT(в форматі непрямого кольору).

Паралельно з процесом перетворення кольору, значення альфа можуть бути додані або зміненізалежно від величини, встановленої в реєстрі DMA2D_FGPFCCR. Якщо вихідне зображення не має альфа-канал, значення альфа–0xFF за замовчуванням автоматично додається, щоб отримати повністю

непрозорий піксель. Значення альфа може змінюватись відповідно до AM[1:0] бітів регістра DMA2D_FGPFCCR:

• може бути збереженим таким який він є (ніяких змін),

• може бути замінене на значення, вказане взначенні ALPHA [7: 0] регістра DMA2D_FGPFCCR,

• може бути замінене вихідним значенням альфа-каналу, помноженим на значення ALPHA [7: 0] регістрів DMA2D_FGPFCCR / DMA2D_BGPFCCR і розділеним на 255.

Отримані 32-розрядні дані шифруються за допомогою OUT PFC в форматі, визначеному в поліCM [2:0] регістра DMA2D_OPFCCR. Формат виведення пікселя не може бути в непрямому режимі, так як процес генерації CLUT не підтримується. Оброблені дані записуються в комірки пам'яті призначення, що вказуються в регістріDMA2D_OMAR.

6. Пам'ять-пам'ять з PFC і змішування

В цьому режимі, два джерела отримуються з FGi BG FIFO з комірок пам'яті, щовизначені регістрами DMA2D_FGMAR і DMA2D_BGMAR. Два перетворювачі формату пікселів повинні бути налаштовані, як описано в режимі пам'ять-пам'ять. Їх конфігурації можуть відрізнятися, так як кожен конвертор формату пікселя єнезалежним і має свою власну пам'ять CLUT. Після того, як кожен піксель був перетворений в 32-бітний за допомогою їх відповідних PFCs, вони змішуютьсявідповідно до формул (1,2).

Отримане 32-бітне значення пікселя шифрується виходом PFC відповідно до заданого вихідного формату, а дані записуються в комірки пам'яті призначення, що вказуються в регістрі DMA2D_OMAR.

7. Контролер зовнішньої пам'яті Flexible memory controller (FMC).

До появи STM32F4x9 сімейство STM32F4 було обладнане контролером зовнішньої статичної пам'яті FSMC. FSMC забезпечував обмін даними між шиною AHB і контролерами пам'яті: NOR Flash / PSRAM, NAND Flash / PC

Card. В STM32F4x9 замість FSMC реалізований FMC. Головною його відмінністю є підтримка SDRAM-пам'яті. Контролер SDRAM-пам'яті має низку особливостей:

• Підтримка двох банків SDRAM з незалежною конфігурацією (адресація до 512 Мб).

• 8/16/32-бітна шина даних.

• Частота тактування HCLK / 2,HCLK / 3 (HCLK може бути до180 МГц).

• Програмовані таймінги.

• Автоматична регенерація з можливістю завдання періоду оновлення.

• Режими зниженого споживання: режим самооновлення і режим Power-down.

Підтримка SDRAM розширює можливості STM32F4 в областях з високими вимогами до пам'яті, наприклад: графічні додатки, мультимедійні додатки і ін.

Основні особливості FMC.

Функціональний блок FMC виконує інтерфейс з синхронною і асинхронноюстатичною пам’ятю, SDRAM пам'ятю та 16-бітними картами пам'яті PC. Його основними завданнями є- переведення AHB транзакцій у відповідному протоколі зовнішнього пристрою, відповідно до часових вимог доступу зовнішніх запам'ятовуючих пристроїв.

Всі зовнішні запам'ятовуючі пристрої розшарюють сигналиадрес, даних і керуючих сигналів з контролером. Кожензовнішньій пристрій, доступний за допомогою унікального вибору кристала. FMC виконуєтільки один доступ одночасно до зовнішнього пристрою.

Додаткові особливості контролера FMC:

• Інтерфейс з синхронною DRAM (SDRAM / Mobile LPSDR SDRAM) пам'ятю

• Підтримка режиму серійної зйомки для більш швидкого доступу до синхронних пристроїв, таких як NOR Flash пам'ять(PSRAM і SDRAM)

• Програмований тривалий вихідний лічильник для асинхронних і синхронних доступів

• Незалежне Chip Select управління для кожного банку пам'яті

• Незалежна конфігурація для кожного банку пам'яті

• Вибір виходів для використання з PSRAM, SRAM і SDRAMприладів

• Зовнішне асинхронне управління очікуванням

• Запис даних FIFO з глибиною 16 33-біти

• Запис адреси FIFO з глибиною 16x30-бітів

• Зчитування FIFO з глибиною 6 x32-біти та 6 x14-бітів адреси тегів) для SDRAM контролера.

FMC має два блоки кеш-пам’яті Write FIFOs: FIFOWriteData з глибиною16x33-біти і WriteAddress FIFO з глибиною16x30бітів.

• WriteData FIFO зберігає дані AHB записуючи в пам'ять (до 32 біт) плюс один біт для передачі AHB (непослідовнийрежим)

• WriteAddressFIFO зберігає адресу AHB (до 28біт) плюс розмір AHB даних (до2 біти). При роботі в режимі серійної зйомки, тільки початкова адреса зберігається винятково при перетині кордону сторінки (для PSRAM і SDRAM). Вцьому випадку, AHB розбивається на два FIFO.

При запуску контакти FMC повинні бути налаштовані додатком для користувача. Контакти FMC I/O, якіне використовуються додатком можуть бути використані для інших цілей. Регістри FMC, які визначають зовнішній вигляд пристрою і пов'язані характеристиками зазвичай встановлюється під

час завантаження і не міняються до наступного скидання або включення живлення. Проте,установки можуть бути змінені в будь-який час.

Структурна схемаFMC наведенанарис.9.6.

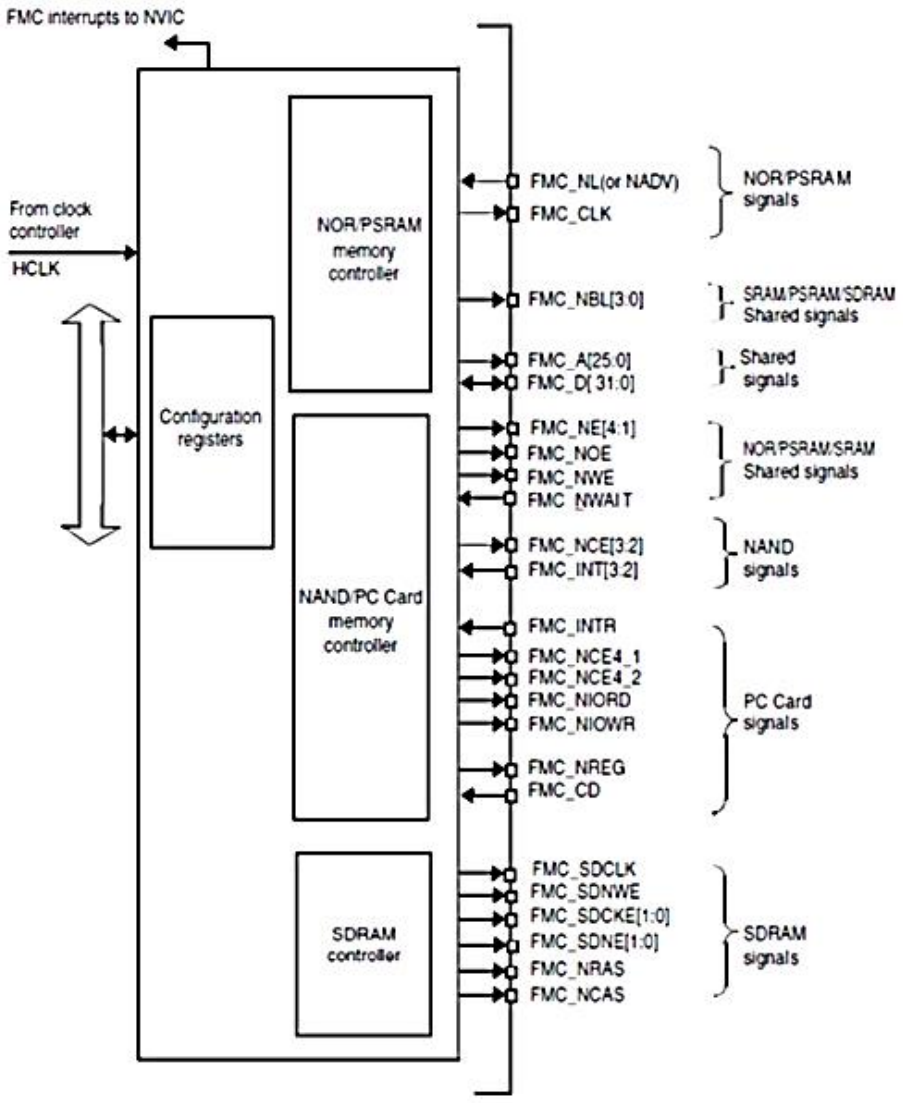


Рис.9.6. Структурна схема FMC

Структурна схема FMC складається зп'яти основнихблоків:

•Інтерфейс AHB (у томучислірегістрівконфігурації FMC)

• NORFlash / PSRAM / SRAM контролер

•Контролеркарти NAND Flash/PC

•Контролер SDRAM

•Інтерфейсзовнішнього пристрою

8. Відображення адреси внутрішнього пристрою

Зточкизору FMC, зовнішняпам'ятьрозділенана6банківфіксованогорозмірупо 256Мбайткожен(рис.9.7):

•Банк1 використовуєтьсядля вирішеннядо 4 NOR флеш-пам'яті або PSRAM пристроїв. Цей банк розділений на 4 NOR / PSRAM підбанки з вибором 4 виділених мікросхем NOR / PSRAM (i=1-4).

•Банки2 і3використовуютьсядля адресуванняпристроїв NAND флеш-пам'яті (1 пристрійв банку)

•Банк4 використовуєтьсядля адресування PCCard

•Банк5і 6використовуютьсядля адресування SDRAM пристров (1пристрійв банку).

Кожен банк пам'яті, який буде використовуватися, може бути налаштований за допомогою користувальницького додаткачерез регістр конфігурації.

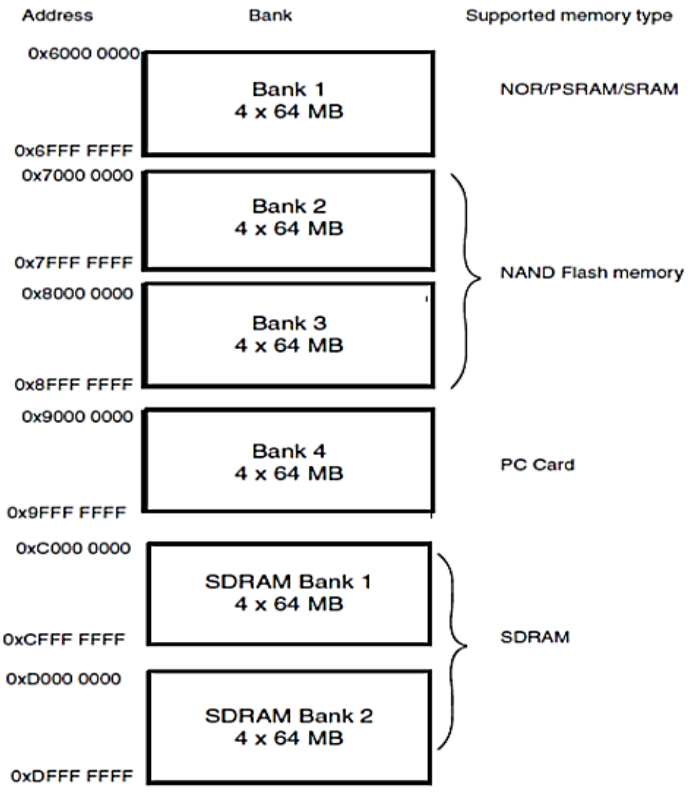


Рис.9.7. Банки пам'яті FMC

Mode 1 - SRAM/PSRAM (CRAM)

Нарис.9.8, 9.9 показані часові діаграми транзакційзчитування і запису даних для підтримуваних режимів з наступним вимогами конфігурацій FMC _BCRx і FMC_BTRx / FMC_BWTRx регістрів.

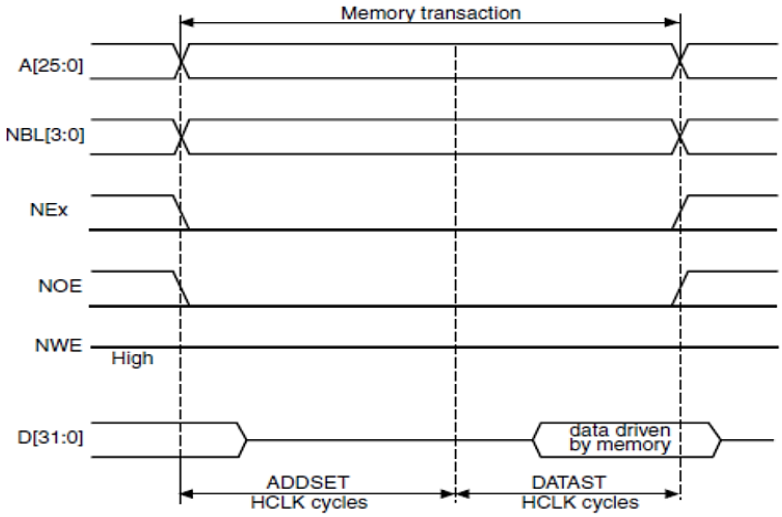


Рис.9.8. Режим 1 зчитування сигналів доступу

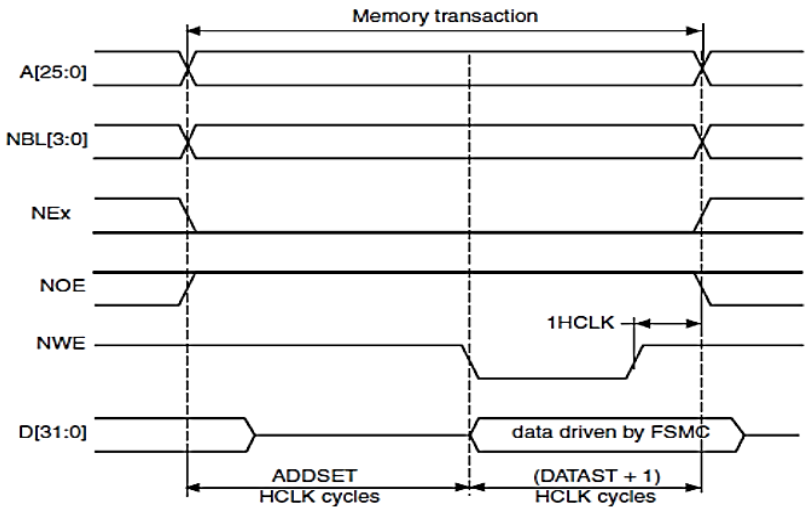


Рис.9.9. Режим 1 запису сигналів доступу

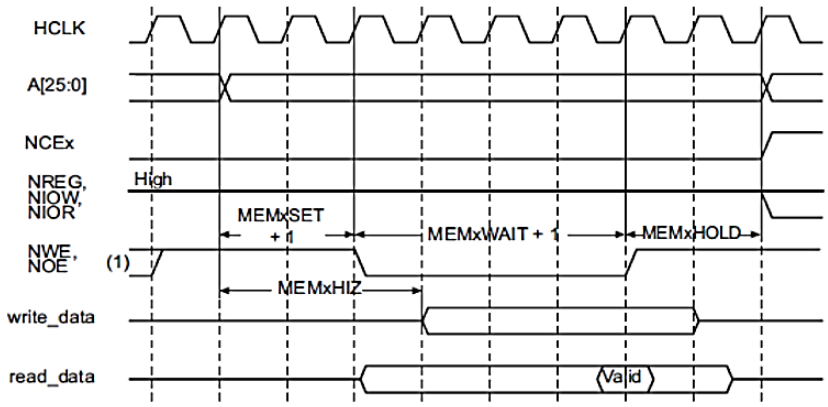


Рис.9.10. NAND Flash/PC Card контролер сигналів длязагального доступу пам'яті

Тема 3.10 Ethernet (ETH): керування доступом до середовища MAC (media access control) з контролером DMA

Периферійний Ethernet в STM32F4xx призначений для передачі і прийому даних через Ethernet відповідно до стандарту IEEE 802.3-2002.

Ethernet забезпечує налаштовуваний, гнучкий периферійний пристрій для задоволення потреб клієнтів та різних додатків. Він підтримує два стандартних інтерфейси до зовнішнього фізичного рівня (PHY): за замовчуванням медіа-незалежний інтерфейс (MII) з визначеними у IEEE 802.3 характеристиками та розділений медіа - незалежний інтерфейс (RMII). Він може бути використаний у якості додатків, таких як комутатори, мережеві карти і т.д.

Ethernet відповідає наступним стандартам:

• IEEE 802.3-2002 для Ethernet MAC //до 1Гбіт

• IEEE 1588-2008 для точної мережевої синхронізації часу

• AMBA 2.0 для портів AHB Master / Slave

• RMII специфікація від RMII консорціуму

Основні особливості Ethernet

Периферійний Ethernet (ETH) має наступні характеристики, що розділяються за категоріями особливостей MAC, DMA, PTP:

Основні особливості MAC

• Підтримка швидкості передачі даних 10/100 Мбіт/с з зовнішніх інтерфейсів

• IEEE 802.3-сумісний інтерфейс MII для зєднання із зовнішнім Fast Ethernet PHY

• Підтримка дуплексних та напів-дуплексних операції:

- Підтримка протоколу CSMA/CD для напів-дуплексного режиму
- Підтримка контролю потоку IEEE 802.3x для дуплексного режиму

- Додаткова пересилка отриманих контрольних кадрів паузи в користувальницький додаток в дуплексних операціях
- Підтримка зворотного зв’язку для напів-дуплексного режиму

• Преамбула старту кадру даних (SFD start-of-frame data) вставка в передачу, і видалення з шляху отримання.

• Програмована довжина кадру для підтримки стандартних кадрів розміром до 16 Кб

• Програмований проміжок між кадрами (40-96 бітові з кроком 8)

• Підтримка різних гнучких режимів фільтрації адреси:

- До чотирьох 48-бітних досконалих (DA) адресних фільтрів з масками для кожного байта
- До трьох адрес порівняльної перевірки 48-бітної адреси джерела з масками для кожного байта
- 64-розрядний хеш-фільтр для групової та односпрямованої (DA) адреси

• Окремий 32-бітний статус повернення до передачі і прийому пакетів

• Підтримка IEEE 802.1Q VLAN виявлення тега для прийому кадрів

• Окремі інтерфейси передачі, прийому та контролю для додатків

• Підтримка обов'язкової мережевої статистики з RMON/MIB лічильників (RFC2819 / RFC2665)

• Інтерфейс MDIO для конфігурації та керування пристроєм PHY

• Виявлення LAN WakeUp кадрів і AMD Magic Packet кадрів

• Функція отримання контрольної суми без навантаження для отриманих IPv4 і TCP-пакетів що інкапсулюється в кадр Ethernet

• Функція покращеного отримання для перевірки контрольної суми заголовка IPv4 і TCP, UDP, ICMP або контрольної суми інкапсульованої в IPv4 або IPv6 датаграм

• Підтримка Ethernet кадрового штампування часу, як описано в IEEE 1588-2008. Шістдесят чотирьох бітні відмітки часу в режимі передачі кожного кадру або отримання статусу

• Два комплекти FIFO: 2-KB передачі FIFO з програмованим порогом можливостей, та 2-KB прийом FIFO з налаштовуваного порога (за замовчуванням 64 байт)

• Можливість фільтрувати всі кадри помилки на прийомі, а не направляти їх в Режим збереження

• Можливість відправляти хороші кадри малого розміру

• Підтримка статистики по генерації імпульсів для кадрів, пропущених або пошкоджених (у зв'язку з переповнення) в приймальному FIFO

• Підтримка зберігати і пересилати механізм для передачі ядра MAC

• Автоматична генерація контрольної паузи кадру або сигналу зворотного зв’язку до ядра MAC на основі отриманого рівня FIFO-заповненості

• Ручне та автоматичне пересилання колізійних кадрів при передачі

• Програмне керування Tx FIFO

• Обчислення і вставка заголовку контрольної суми IPv4 і TCP, UDP.

• Підтримка внутрішнього шлейфу на MII для налаштування

1. Можливості DMA

• Підтримка всіх AHB типів розриву в AHB Slave інтерфейсі

• Програмне забезпечення може вибрати тип AHB розриву (фіксований або невизначений розрив) в Master AHB інтерфейсі.

• Оптимізація для пакетно-орієнтованих DMA пересилань з кадровим розділенням

• Байт вирівнювання адресований для підтримки буфера даних

• Дескриптор ланцюжка: подвійний-буферний або пов'язаний список

• Дескриптор архітектури, дозволяє передавати великі блоки даних з мінімальним втручанням CPU

• Кожен дескриптор може передавати до 8 Кбайт даних

• Комплексний статус звітності для нормальної роботи і передачі з помилками

• Індивідуальний програмований розмір пакета для прийому і передачі DMA для оптимального використання шини

• Програмовані параметри переривань для різних умов експлуатації

• Повний контроль переривань покадрової передачі/прийому

• Круговий або фіксованою пріоритет арбітражу між прийомом та передачею

• Режими Start/Stop

• Поточний Tx / Rx Покажчик на буфер, як регістрів стану

• Поточний Tx / Rx дескриптор курсору, регістрів стану

Можливості PTP (Precision Time Protocol)

• Отримання та передача кадрів фіксації часу

• Грубі та точні методи корекції

• Trigger переривання, коли час системи стає більшим, ніж цільової час

• Вихід імпульс за секунду

2. Порти Ethernet

Таблиця 10.1 показує сигнали MAC і відповідне відображення сигналу MII / RMII. Усі сигнали MAC надходять на AF11, деякі сигнали надходять на різні порти вводу/виводу, і повинні бути налаштовані в режимі альтернативної функції.

Таблиця 10.1

Port	**AF11**
	ETH
PA0-WKUP	ETH_MII_CRS
PA1	ETH_MII_RX_CLK / ETH_RMII_REF_CLK
PA2	ETH_MDIO
PA3	ETH_MII_COL
PA7	ETH_MII_RX_DV / ETH_RMII_CRS_DV
PB0	ETH_MII_RXD2
PB1	ETH_MII_RXD3
PB5	ETH_PPS_OUT
PB8	ETH_MII_TXD3
PB10	ETH_MII_RX_ER
PB11	ETH_MII_TX_EN / ETH_RMII_TX_EN
PB12	ETH_MII_TXD0 / ETH_RMII_TXD0
PB13	ETH_MII_TXD1 / ETH_RMII_TXD1
PC1	ETH_MDC
PC2	ETH_MII_TXD2
PC3	ETH_MII_TX_CLK
PC4	ETH_MII_RXDO / ETH_RMII_RXD0
PC5	ETH_MII_RXD1 / ETH_RMII_RXD1
PE2	ETH_MII_TXD3
PG8	ETH_PPS_OUT
PG11	ETH_MII_TX_EN / ETH_RMII_TX_EN
PG13	ETH_MII_TXD0 / ETH_RMII_TXD0
PG14	ETH_MII_TXD1 / ETH_RMII_TXD1
PH2	ETH_MII_CRS
PH3	ETH_MII_COL
PH6	ETH_MII_RXD2
PH7	ETH_MII_RXD3
PI10	ETH_MII_RX_ER

Функціональний опис Ethernet: SMI (Station management interface), MII (Media-independent interface) і RMII (Reduced media-independent interface)

Периферійний Ethernet складається з MAC 802.3 з виділеним контролером DMA. Він підтримує як медіа-незалежний інтерфейс за замовчуванням (MII) та розділений медіа-незалежний інтерфейс (RMII) через вибір одного біта (SYSCFG_PMC регістр). Контролер DMA взаємодіє з ядром та пам’яттю через AHB Master та Slave інтерфейси. AHB Master інтерфейс контролює передачу даних в той час як інтерфейс AHB Slave має доступ до простору регістра контролю та статусу (CSR). Дані, які потрібно передати з буферу FIFO (Tx FIFO), зчитуються з пам'яті по DMA перед передачею по MAC Core. Аналогічним чином, прийом FIFO (Rx FIFO) зберігає кадри Ethernet, отримані від лінії, поки вони не будуть передані системній пам’яті по периферії DMA Ethernet також включає в себе SMI для зв'язку із зовнішнім PHY. Набір регістрів конфігурації дозволяють користувачеві вибрати потрібний режим і особливості для MAC і контролером DMA.

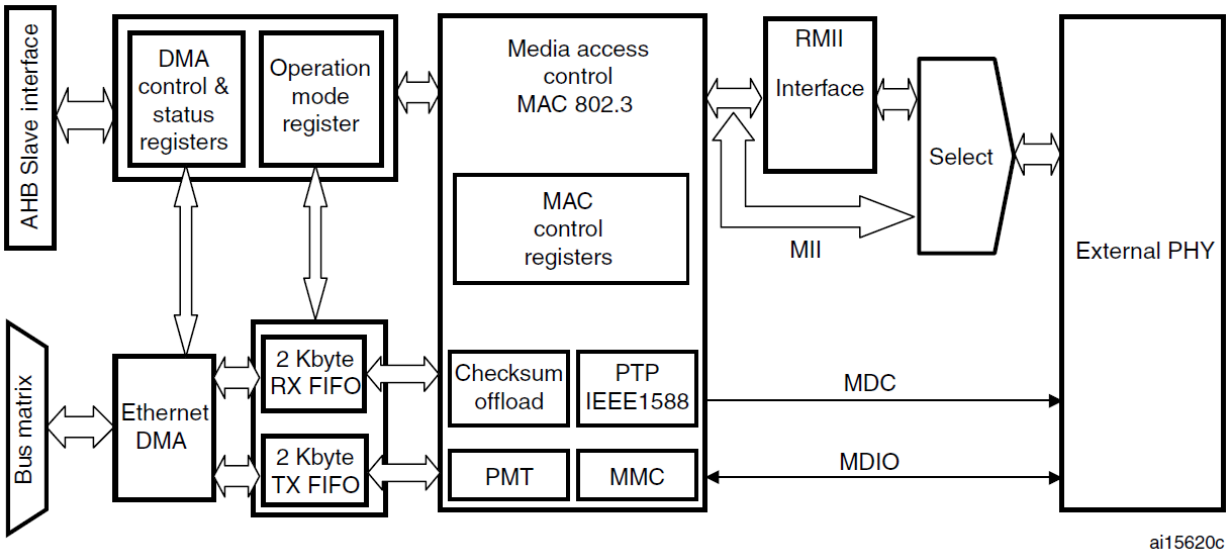


Рис.10.1 Блок схема Ethernet

Блок схема включає наступні блоки:

- інтерфейс AHB (AHB slave interface)

- матриця шин (Bus matrix)
- контролер DMA (Ethernet DMA)
- регістр контролю та статусу DMA (DMA control & status register)
- регістр режиму роботи (Operation mode register)
- буфер FIFO для запису та зчитування (TX, RXFIFO)
- блок керування доступом до середовища (MAC 802.3):
- регістри контролю MAC (MAC control registers)
- блок розвантаження контрольної суми (Checksumoffload)
- протокол точного часу (PTPIEEE 1588)
- лічильники керування MAC (MMC – MAC management counters)
- блок керування енергоспоживання MAC (PMT – powermanagement)
- інтерфейс RMII (RMII Interface)
- інтерфейс MII (MII)
- блок вибору робочого інтерфейсу (Select)
- зовнішній фізичний рівень (External PHY)

3. Інтерфейс керування станції: SMI (Station management interface)

Інтерфейс керування станцією (SMI) дозволяє додатку отримати доступ до будь-яких регістрів PHY через 2-провідну лінію синхронізації і даних. Інтерфейс підтримує доступ до 32 PHYs.

Додаток може вибрати один з 32 PHYs і один з 32 регістрів в рамках якої-небудь PHY і відправити дані керування або отримувати інформацію про статус. Тільки один регістр в одному PHY може бути змінений в будь-який момент часу.

Обидві лінії генератора тактових імпульсів (MDC) та даних (MDIO) реалізовані у вигляді альтернативної функції вводу/виводу в мікроконтролері:

• MDC: періодичний ГТІ (генератор тактових імпульсів), який надає посилання синхронізації для передачі даних на максимальна частота 2,5 МГц.

Мінімально високі і низькі проміжки часу для MDC повинні бути 160 нс кожен, і мінімальний період для MDC повинен бути 400 нс. В неробочому стані інтерфейс управління SMI приводить в MDC мінімальний тактовий сигнал.

• MDIO: введення/виведення даних бітового потоку для передачі інформації про стан в або з PHY пристрою синхронно з тактовим сигналом MDC

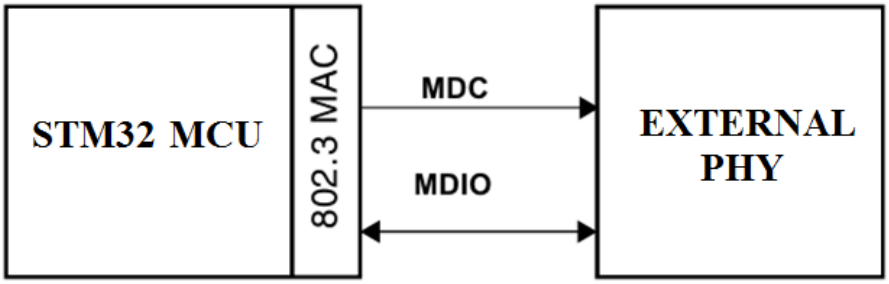


Рис. 10.2 Сигнали інтерфейсу SMI

4. Формат кадру SMI

Структура кадру, що відноситься до операції читання або запису, показано в таблиці 2, порядок передачі бітів повинен бути зліва направо.

Таблиця 10.2

	Management frame fields							
	Preamble (32 bits)	Start	Operation	PADDR	RADDR	TA	Data (16 bits)	Idle
Read	1…1	01	10	ppppp	rrrrr	Z0	dddddddddddddd	Z
Write	1…1	01	01	ppppp	rrrrr	10	dddddddddddddd	Z

Кадр керування складається з восьми полів:

• Preamble: кожна транзакція (читання або запис) може бути ініційований з області преамбули, що відповідає 32 суміжній логіці, один біт на MDIO відповідно до 32 відповідних циклів на MDC. Це поле використовується для встановлення синхронізації з пристроєм PHY.

• Start: початок кадру визначається зразком <01> для перевірки переходів на лінії стану логіки від одиниці до нуля і знову до одиниці.

• Operation: визначає тип транзакції (зчитування або запис) на стадії розробки.

• PADDR: адреса PHY становить 5 біт, що дозволяє 32 унікальні адреси PHY. MSB біт адреси є першим що передається і приймається.

• RADDR: адреса регістра 5 біт, що дозволяє 32 індивідуальні регістри можуть бути розглянуті в рамках обраного PHY пристрою. MSB біт адреси є першим що передається і приймається.

• TA: поле розвороту визначає 2-бітову послідовність між полями RADDR та даних, щоб уникнути конкуренції під час транзакції читання. Для транзакції читання контролер MAC керує лініями MDIO 2 бітами TA. PHY пристрій керує станом високого опору на перший біт TA, нульовий біт на другий. Для транзакції запису, контролер MAC водить <10> в поле TA.

• Data: поле даних є 16-бітний. Перший біт що передається і приймається повинен бути 15 біт регістра ETH_MIID.

• Idle: MDIO лінія наводиться в високоімпедансное стан. Всі керування трьома станами повинні бути відключені та PHY`s резистор тримає лінію на логічный одниниці.

5. Операція запису SMI

Коли додаток встановлює біти MII запису та зайнятості (в Ethernet MAC MII адресний регістр (ETH_MACMIIAR)), SMI ініціює операцію запису в регістри PHY з передачі адреси PHY, адреси регістра в PHY, та даних запису (в Ethernet MAC MII регістр даних (ETH_MACMIIDR). Додаток не повинен змінювати вміст адресного регістру MII або регістру даних MII в той час як тривають транзакції. Операції запису в адресний регістр MII або регістр даних MII протягом цього періоду ігноруються, і транзакція закінчується без

помилок. Після завершення операції запису, SMI вказує на це шляхом скидання біта зайнятості.

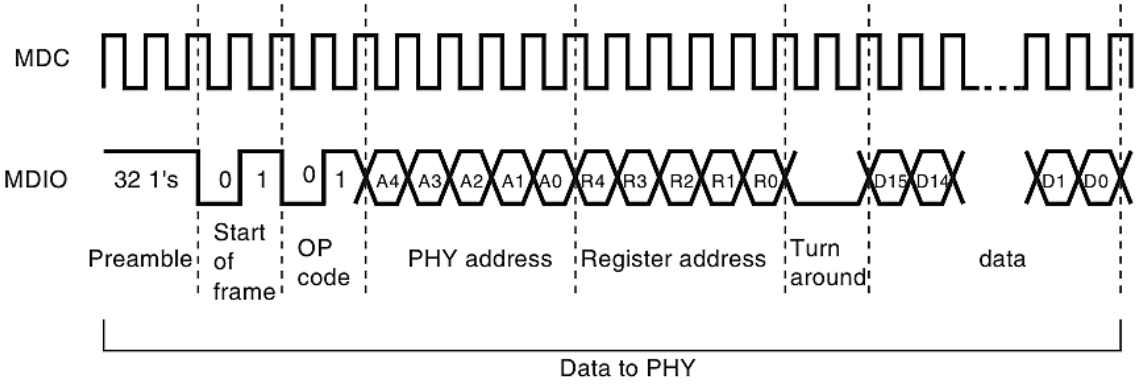


Рис.10.3 MDIO часова структура кадру – Цикл запису

6. Операція зчитування SMI

Коли користувач встановлює біт зайнятості MII в Ethernet MAC адресному регістрі MII (ETH_MACMIIAR) з бітом запису MII на 0, SMI ініціює операцію читання в регістри PHY шляхом передачі адреси PHY та адреси регістра в PHY. Додаток не повинен змінювати адресний регістр MII або регістр даних MII в той час як тривають транзакції. Операції запису в адресний регістр MII або регістр даних MII протягом цього періоду ігноруються, і транзакція закінчується без помилок. Коли операція читання завершена, SMI скидає біт зайнятості, а потім оновлює регістр даних MII даними, зчитанними з PHY.

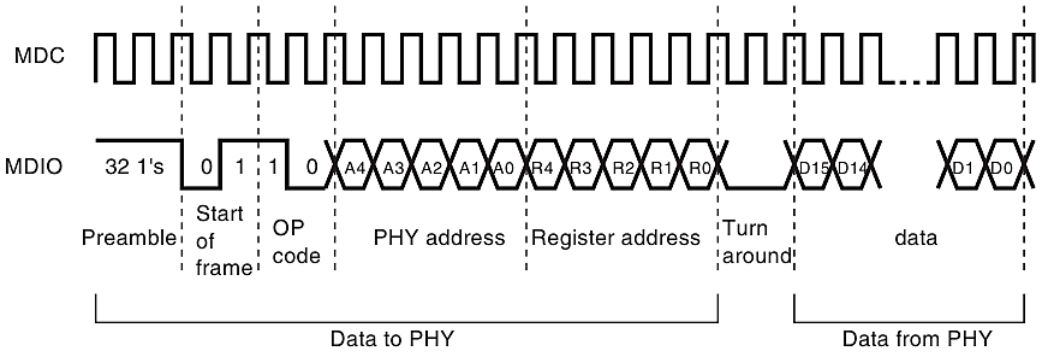


Рис.10.4 MDIO часова структура кадру – Цикл зчитування

7. SMI вибір ГТІ

MAC запускає керування операціями запису/зчитування. У якості SMI генератора тактових імпульсів (ГТІ) виступає AHB генератор. Коефіцієнт поділу залежить від настройки діапазону ГТІ в адресному регістрі MII. Таблиця 10.3 показує, як встановити діапазони частот.

Таблиця 10.3

Selection	HCLK clock	MDC clock
000	60-100 MHz	AHB clock / 42
001	100-168 MHz	AHB clock / 62
010	20-35 MHz	AHB clock / 16
011	35-60 MHz	AHB clock / 26
100, 101, 110, 111	Reserved	-

8. Медіа-незалежний інтерфейс: MII (Media-independent interface)

Медіа - незалежний інтерфейс (MII) визначає взаємозв'язок між підрівнем MAC та PHY для передачі даних в 10 Мбіт/с і 100 Мбіт/с.

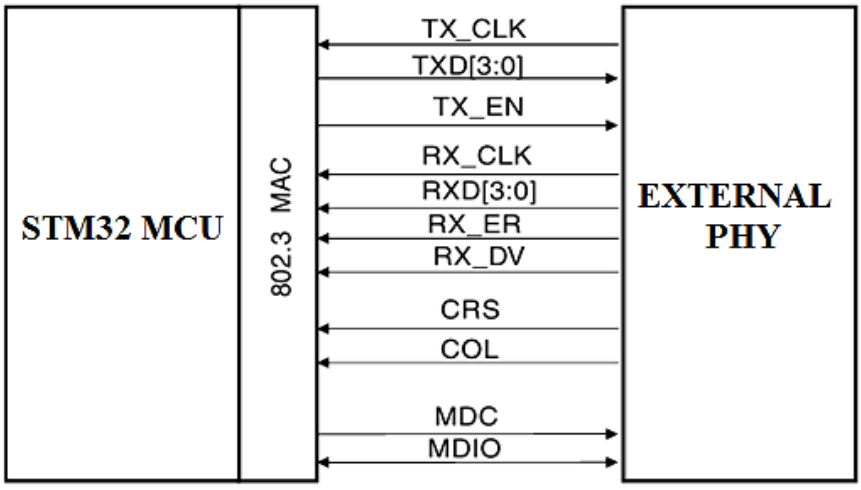


Рис. 10.5 Сигнали інтерфейсу MII

• MII_TX_CLK: безперервний ГТІ, який надає тактові імпульси для передачі даних TX. Номінальна частота: 2,5 МГц при 10 Мбіт/с, 25 МГц на 100 Мбіт/с.

• MII_RX_CLK: безперервний ГТІ, які надає тактові імпульси для передачі даних RX. Номінальна частота: 2,5 МГц при 10 Мбіт/с; 25 МГц на 100 Мбіт/с.

• MII_TX_EN: передача ввімкнена, вказує, що MAC надає MII для передачі. Повинен бути синхронізованим (MII_TX_CLK) з першого байта, преамбули і повинен залишається в активному стані, поки всі дані не будуть передані в MII.

• MII_TXD [3: 0]: передає дані комплектом з 4 сигналів даних ведених синхронно по підрівні MAC і кваліфікуються на затвердження сигналу MII_TX_EN. MII_TXD [0] молодший біт, MII_TXD [3] найстарший біт. Тоді як MII_TX_EN буде скинутий дані що передають не повинні мати вплив на PHY.

• MII_CRS: контроль несучої затверджується PHY, коли передача або прийом в режимі очікування. Він повинен бути скинутий за допомогою PHY, коли передача або прийом в режимі очікування.

• MII_COL: виявлені колізії повинні бути заявлені в PHY при виявленні колізій на носії, і повинні бути затвердженими тоді як стан колізії зберігається.

• MII_RXD [3: 0]: дані прийому являє собою комплект з 4 сигналів даних ведених синхронно по PHY і кваліфікованих на затвердження сигналу MII_RX_DV. MII_RXD [0] є молодший біт, MII_RXD [3] є найстарший біт. Тоді як MII_RX_EN буде скинутий і MII_RX_ER затверджується, специфічні значення MII_RXD [3: 0] використовується для передачі певної інформації від PHY.

• MII_RX_DV: отримання дійсних даних

• MII_RX_ER: виявлення помилки.

Таблиця 10.4 TX інтерфейс, кодування сигналу

MII_TX_EN	MII_TXD[3:0]	Description

0	0000 through 1111	Normal inter-frame
1	0000 through 1111	Normal data transmission

Таблиця 10.5 RX інтерфейс, кодування сигналу

MII_TX_EN	MII_RX_ERR	MII_TXD[3:0]	Description
0	0	0000 through 1111	Normal inter-frame
0	1	0000	Normal inter-frame
0	1	0001 through 1101	Reserved
0	1	1110	False carrier indication
0	1	1111	Reserved
1	0	0000 through 1111	Normal data reception
1	1	0000 through 1111	Data reception with errors

9. MII вибір ГТІ

Щоб генерувати тактові сигнали TX_CLK та RX_CLK, зовнішній PHY повинен бути синхронізований з зовнішнім тактовим генератором 25 МГц. Замість використання зовнішнього тактового генератора 25 МГц STM32F4xx microcontroller може виводити цей сигнал на MCO пін. В цьому випадку, множник PLL повинен бути налаштований таким чином, щоб отримати бажану частоту на піні MCO, від 25 МГц зовнішнього генератора.

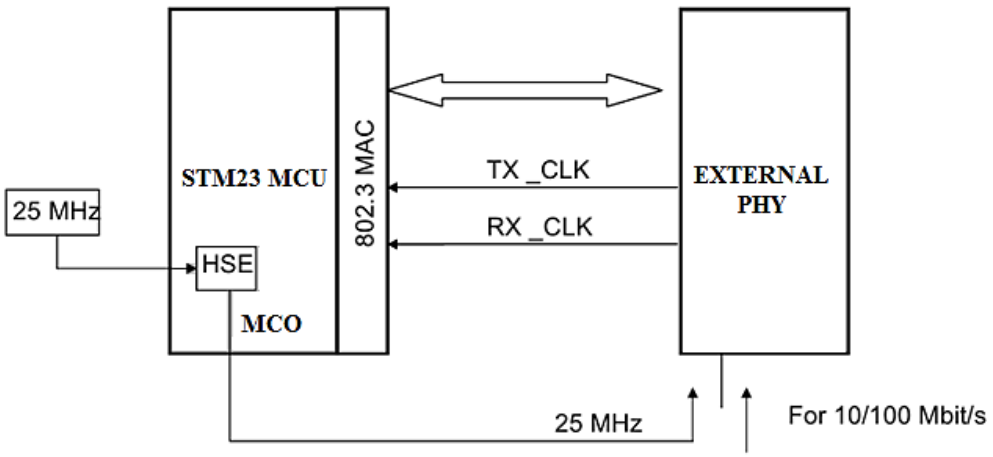


Рис. 10.6

10. Розділений медіа незалежний інтерфейс: RMII (Reduced media-independent interface)

Розділений медіа- незалежний інтерфейс (RMII) зменшує число виводів між мікроконтроллерним переферійним Ethernet та зовнішнім Ethernet 10/100 Мбіт/с. Відповідно до стандарту IEEE 802.3u, MII містить 16 контактів для передачі даних і контролю. Специфікація RMII присвячена зменшити кількість контактів до 7.

RMII блок має наступні характеристики:

• Він підтримує 10Мбіт/с і 100 Мбіт/с діяльність

• Частота опорного ГТІ повинна бути в два рази більшою 50 МГц

• Такі ж опорної частоти повинні бути отримані ззовні, як для MAC так і зовнішнього Ethernet PHY

• Він забезпечує незалежний 2-бітний широкий тракт передачі і прийому даних

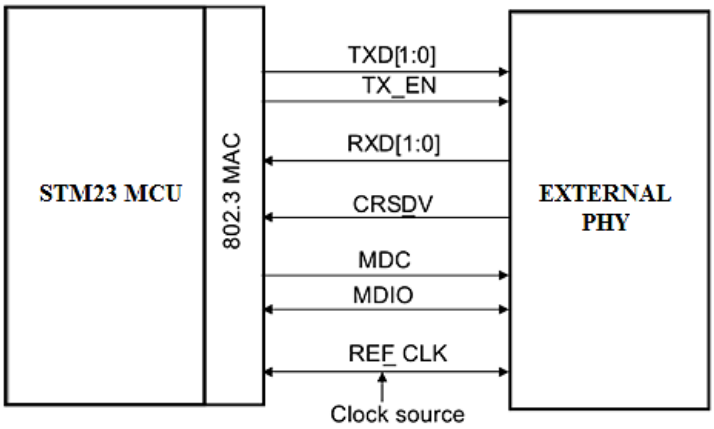


Рис.10.7 Сигнали інтерфейсу RMII

11. RMII вибір ГТІ

Можна використовувати зовнішній генератор на 50 МГц, або використовувати ГТІ мікроконтроллера з додатковим використанням PLL для отримання частоти 50 МГц.

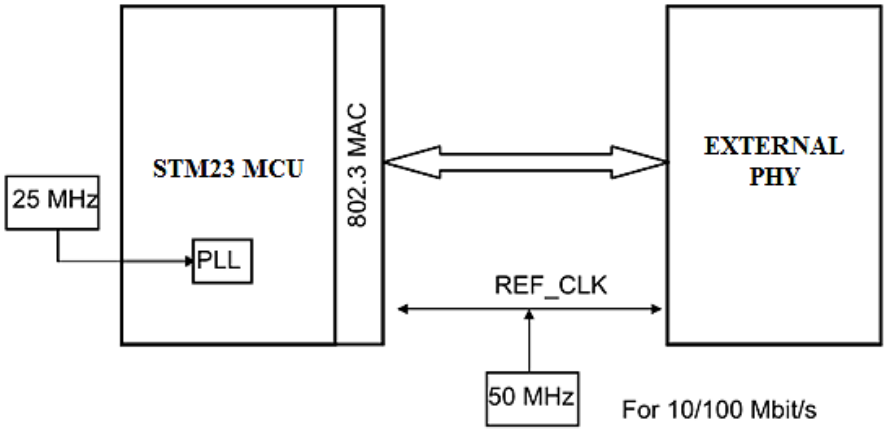


Рис. 10.8

Вибір MII/RMII

Режим, MII або RMII, вибирається за допомогою біт конфігурації 23, MII_RMII_SEL, в регістрі SYSCFG_PMC. Додаток має встановити режим MII /RMII тоді як контролер Ethernet знаходиться в режимі скидання або перед включенням тактових імпульсів.

MII/RMII внутрішня схема генератора тактових імпульсів

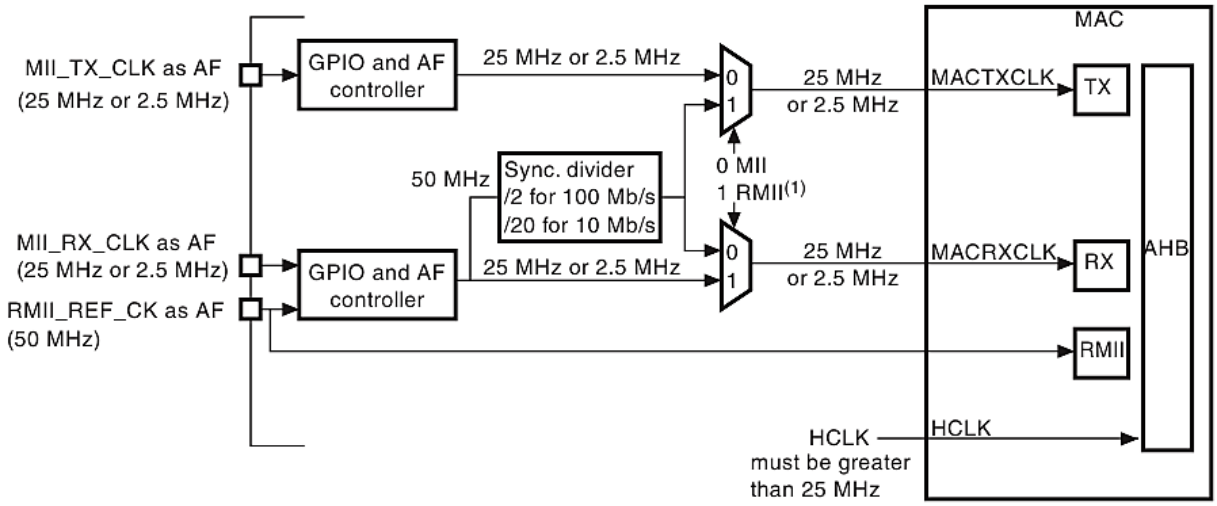


Рис. 10.9 Внутрішня схема ГТІ

Внутрішня схема ГТІ рис. 10.9 включає:

- контролер GPIO та AF (GPIO and AF controller)
- синхро драйвер (Sync. Driver)
- блоки вибору робочого інтерфейсу (0 1)
- контролер MAC (MAC)

Ethernet функціональний опис: MAC 802.3

IEEE 802.3 Міжнародний стандарт для локальних мереж (LAN) працює CSMA/CD в якості методу доступу.

Периферійний Ethernet складається з MAC 802. контролера з медіа незалежним інтерфейсом (MII) і спеціальним контролером DMA.

Підрівень MAC виконує такі функції:

• інкапсуляція даних (передача і прийом)

- Обрамлення (кордон кадру, кадрова синхронізація)

- Рішення (обробка адрес джерела та пункту призначення)

- Виявлення помилок

• Керування доступом до середовища

- Розподіл середній (попередження колізій)

- Дозвіл розбіжностей (обробка колізій)

MAC 802.3 формат кадрів

Існує два формати кадрів для систем передачі даних з використанням CSMA/CD MAC:

• Базовий формат кадру MAC

• Мічений формат кадру MAC (розширення базового формату кадру MAC)

Рис. 10.11 та рис. 10.12 описують структури кадрів (без тегів і з міткою), які включають наступні поля:

• Preamble: поле 7-байт використовується для синхронізації (PLS схема). Шістнадцяткове значення: 55-55-55-55-55-55-55

Біт картина: 01010101 01010101 01010101 01010101 01010101 01010101 01010101 (справа-наліво біти передачі)

• Start frame delimiter (SFD): поле 1 байт використовується для позначення початку кадру.

Шістнадцяткове значення: D5

Біт картина: 11010101 (справа-наліво біти передачі)

• Destination and Source Address fields: 6-байтових полів для вказівки місця призначення і джерела, станція звертається таким чином (рис 10):

- Кожна адреса 48 біт в довжину
- Перший LSB біт (I/G) в полі адреси призначення використовується для позначення індивідуальний (I / G = 0), або групової адреси (I/G = 1). Групова адреса може визначити жодної, одну, декілька, або всі станції, підключені до локальної мережі. Адреса джерела - перший біт зарезервований і скинутий в 0.
- Другий біт (U/L) розрізняє локальні (U/L = 1) або глобальні (U / L = 0) адреси. Для широкополосних адрес цей біт також 1.

- Кожен байт кожного поля адреси повинен бути передані молодший біт перший.

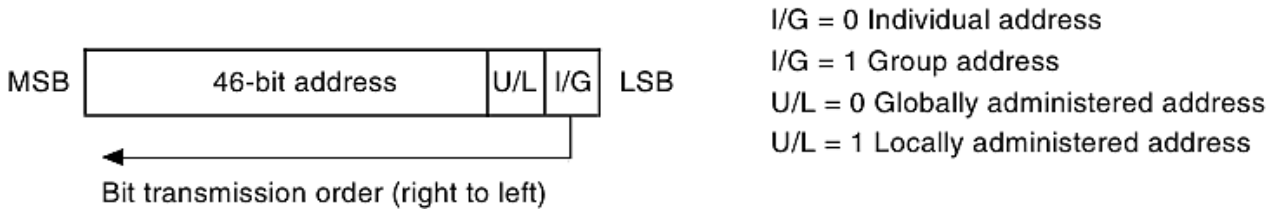


Рис. 10.10 Формат поля адреси

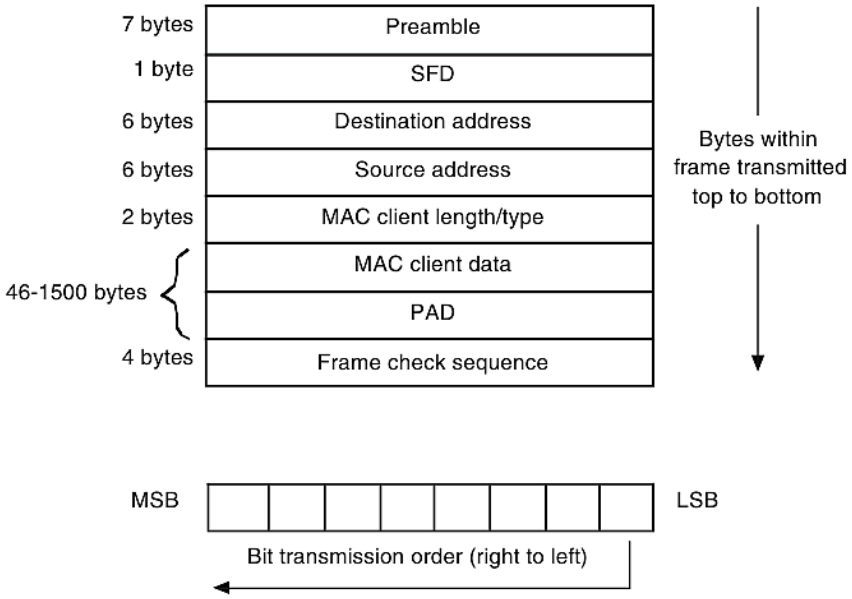


Рис. 10.11 Базовий формат кадру MAC

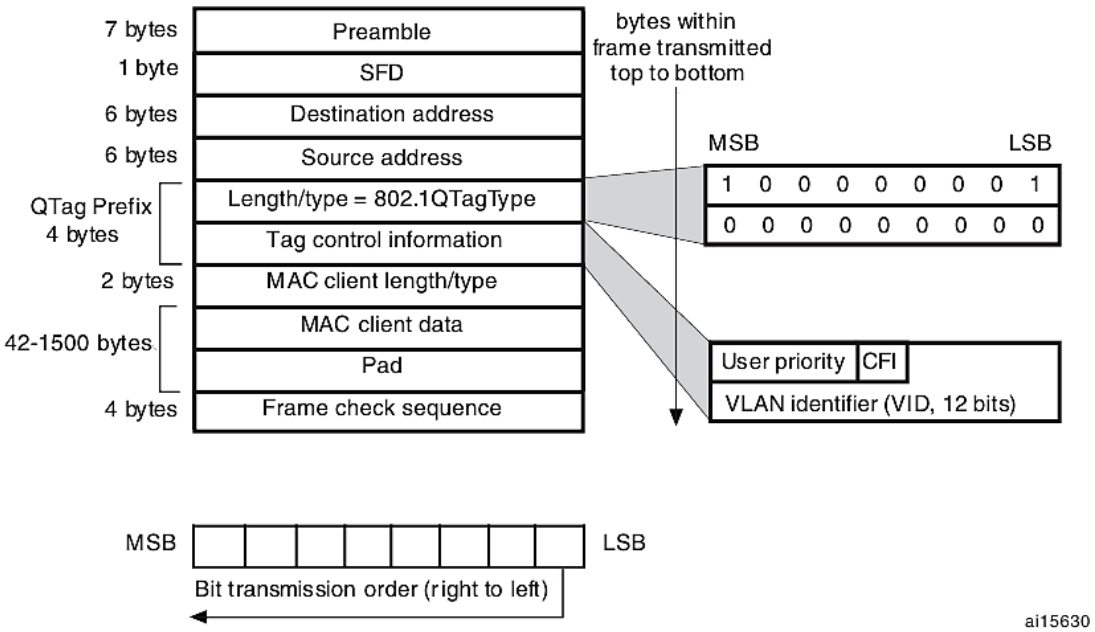


Рис. 10.12 Мічений формат кадру MAC

12. Передача кадру MAC

DMA контролює всі транзакції по шляху передачі. Ethernet кадри, зчитані з системної пам'яті відправляються в FIFO по DMA. Кадри потім переносяться в ядро MAC. Коли кінець кадру передається, статус передачі приймається від ядра MAC і переносяться назад у DMA. Передача FIFO має глибину 2 Кбайт. Рівень заповнення FIFO вказується в DMA, так що він може ініціювати вибірку даних з системної пам'яті, використовуючи інтерфейс AHB. Дані з AHB Master інтерфейсу проштовхуються в FIFO.

При виявленні SOF, MAC приймає дані і починає передавати на MII. Час, необхідний для передачі даних кадрів на MII після застосування ініціює передачу, яка змінюється в залежності від факторів затримки, як IFG затримка, час для передачі преамбули/SFD, а також будь-якими затримкам напівдуплексного режиму. Після того як EOF переданий до ядра MAC, ядро завершує нормальну передачу, а потім дає статус передачі назад у DMA.

Є два режими роботи запису даних до ядра MAC:

• Режим порогу, як тільки число байтів в FIFO перетинає пороговий рівень, дані готові до передачі в ядро MAC. Пороговий рівень налаштовується за допомогою TTC біти ETH_DMABMR.

• Режимі зберігання і вперед, тільки повний кадр зберігається в FIFO, кадр відправляється до ядра MAC. Якщо розмір Tx FIFO менший, ніж кадр Ethernet, які повинні передаватися, то кадр передається до ядра MAC, коли Tx FIFO стає майже повним.

13. Передача протоколу

MAC керує роботою передачі кадру Ethernet. Він виконує наступні функції згідно специфікації IEEE 802.3/802.3z:

• генерує преамбулу та SFD

• генерує джем шаблон в напівдуплексному режимі

• контролює таймаут Jabber

• контролює потік для напівдуплексного режиму (зворотній зв'язок)

• генерує статус передачі кадру

• містить тимчасову логіку кадрів відповідно до IEEE 1588

Коли запрошується нова передача кадру, MAC посилає преамбулу і SFD, а потім дані. Преамбула визначається 7 байтами як 0b10101010, SFD визначається 1 байтом як 0b10101011. Вікно колізії визначається як 1 часовий інтервалу (512 бітових раз для 10/100 Мбіт/с Ethernet).

В режимі MII, якщо колізія відбувається в будь-який час з початку кадру в кінці поля CRC, MAC посилає 32-бітний джем шаблон 0x5555 5555 на MII, для повідомлення всіх інших станції що відбулася колізія. Якщо колізія відбулася протягом фази передачі преамбули, MAC завершує передачу преамбули і SFD, а потім відправляє джем шаблон.

Пересилання при колізії

Коли відбулась колізія, MAC вказує на повтору спробу даючи на це статус, ще не отримавши кінця кадру. Тоді вмикається пересилання і кадр відправляється знову з FIFO. Після того як більш ніж 96 байт були відправлені до ядра MAC, контролер FIFO звільняє простір і робить його доступним для DMA. Це означає, що повторна передача не може бути можливою після того, як цей поріг перетинається.

MII / RMII часові діаграми передачі

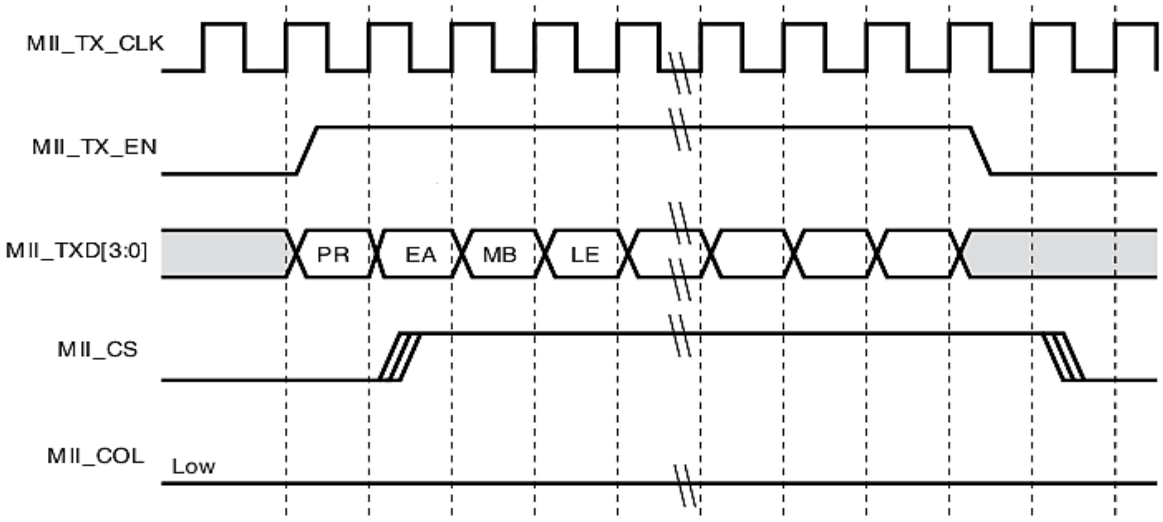


Рис. 10.13 Передача без колізій

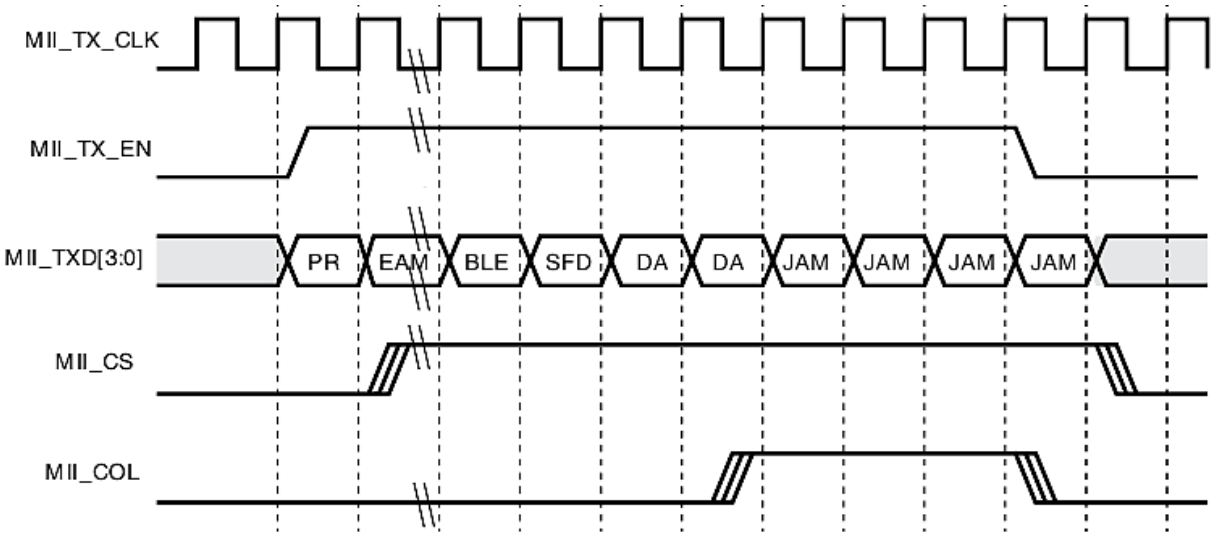


Рис. 10.14 Передача з колізією

На рис. 10.13, 10.14 наведені часові діаграми передачі по MII без колізії та з колізією. Як видно з рис. 10.14 та як було описано раніше – при виниканні колізії під час передачі,MAC завершує передачу преамбули і SFD, а потім відправляє 32-бітний джем шаблон 0x5555 5555 на MII, для повідомлення PHY що відбулася колізія.

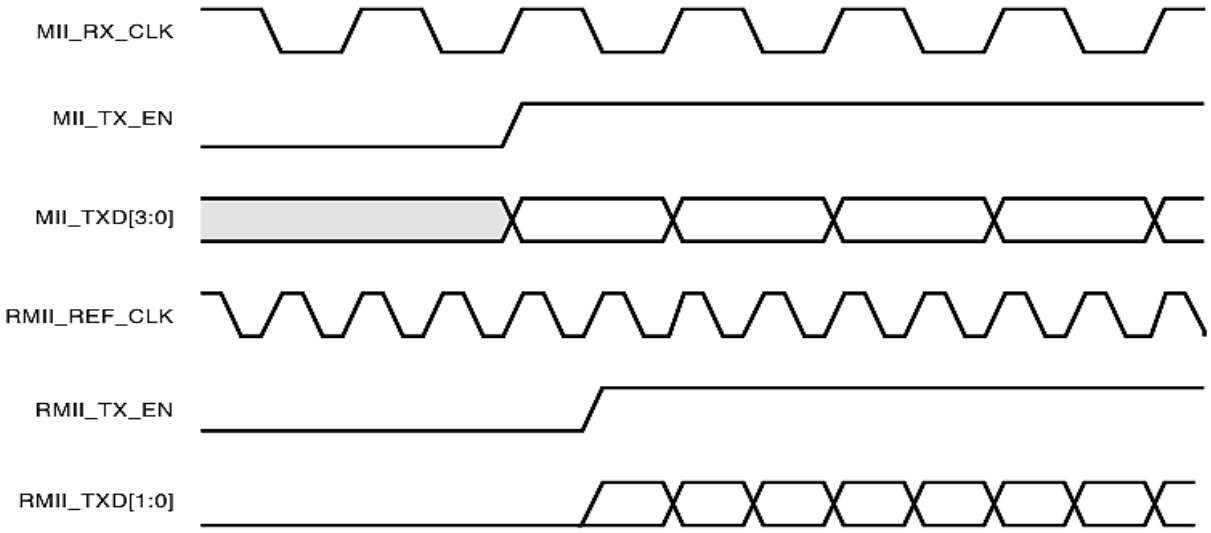


Рис. 10.15 Передача кадрів в MII та RMII режимі

На рис. 10.15 наведені часові діаграми передачі кадрів по MII та RMII. Як видно з рис. 10.15 та як було описано раніше, тактова частота RMII у два рази більша порівняно з тактовою частотою MII.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Мікропроцесорна техніка: Підручник / Ю.І. Якименко, Т.О. Терещенко, Є.І. Сокол, В.Я. Жуйков, Ю.С. Петергеря / За ред. Т.О. Терещенко. – К.: Видавництво “Політехнік”, 2002. – 439 с

2. Гук М. Процессоры Intel: от 8086 до Pentium II – СПб: Питер, 1997. – 224 с.

3. Абель П. Язык Асемблера для IBM PC и программирование / Пер. с англ. Ю.В. Сальникова. – М.: Высш.шк., 1992. – 447 с.

4. Гук М. Процессоры Pentium II, Pentium Pro и просто Pentium – СПб: Питер Ком, 1999. – 288 с.

5. Корнеев В.В., Киселев А.В. Современные микропроцессоры– М. НОЛИДЖ, 2002 – 320 с,

6. Корнеев В.В., Киселев А.В. Современные микропроцессоры. – М. НОЛИДЖ, 2003. – 448 с.

7. Морс С.П., Альберт Д.Д. Архитектура микропроцессора 80286. – М.: Радио и связь. – 1990. – 304 с.Самофалов К.Г., Викторов О.В. Микропроцессоры. - К., Техника, 1989.

8. Микропроцессорный комплект К1810. Справочная книга. /Под ред. Казаринова Ю.М. - М., Высшая школа, 1990.

9. Сташин В.В., Урусов А.В. Проектирование цифровых устройств на однокристальных микроконтроллерах. М., Энергоатомиздат, 1990.

10. Липовецкий Г.П., Литвинский Г.В. и др. Однокристальные микро-ЭВМ. Семейство МК48. Семейство МК51. М., 1992.

11. Даль Е. Cortex-M на примере STM32 [Електронний ресурс] / Евгений Даль – Режим доступу до ресурсу: http://geektimes.ru/post/254730/.

12. RM0090 Reference manual [Електронний ресурс] – Режим доступу до ресурсу: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf.

13. Cortex™ -M4 Devices Generic User Guide [Електронний ресурс] – Режим доступу до ресурсу:

http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf.

14. STM32F4. Внешние прерывания на Си [Електронний ресурс] – Режим доступу до ресурсу: http://teplofizik.diary.ru/p188588391.htm?oam.

15. STM32F4 MCU series [Електронний ресурс] – Режим доступу до ресурсу: www.st.com/web/en/resource/technical/document/reference_manual. pdf.

16. Козаченко В.Ф. Микроконтроллеры: руководство по применению 16-разрядных микроконтроллеров Intel MCS-196/296 во встроенных системах управления. - М.: Издательство ЭКОМ, 1997 - 688 с.

17. Коффрон Дж. Технические средства микропроцессорных систем.: М.:Мир,1983

18. Коган Б.М., Сташин В.В. Основы проектирования микропроцессорных устройств автоматики. -М.:Энергоатомиздат,1987

19. http://www.kaf-pe.ntu-kpi.kiev.ua/ Жуйков В.Я., Терещенко Т.О., Петергеря Ю.С. і ін «Мікропроцесори і мікроконтролери» - Електронний підручник

20. http://www.kaf-pe.ntu-kpi.kiev.ua/ Терещенко Т.О., Петергеря Ю.С., Хохлов Ю.В. Методичні вказівки до виконання лабораторних робіт з курсу „Мікропроцесорна техніка”. Мікроконтролери сімейства STMicroelectronics.

21. http://www.kaf-pe.ntu-kpi.kiev.ua/ Терещенко Т.О., Петергеря Ю.С., Хохлов Ю.В. Методичні вказівки з курсу „Мікропроцесорна техніка". Архітектура мікроконтролерів STMicroelectronics.

22. Мороз А.В., Терещенко Т.О. Дослідження захищеності програмного забезпечення мікроконтролерів за струмом споживання / Технічна електродинаміка. Тематичний випуск „Силова електроніка та енергоефективність". – 2008. – Ч.2 – С.99-102.

23. Валиев К. А. Квантовые компьютеры: надежда и реальность / Валиев К. А., Кокин А. А. — Ижевск : РХД, 2001. — 352 с.

24. stm32f4xx_dac.hhttps://github.com/mikeferguson/stm32/blob/9e27786de4e6df74dc735f3c5667fbe4874e59c5/libraries/STM32F4xx_StdPeriph_Driver/inc/stm32f4xx_dac.h

25. https://github.com/mikeferguson/stm32/tree/9e27786de4e6df74dc735f3c5667fbe4874e59c5/libraries/STM32F4xx_StdPeriph_Driver/inc

26. https://github.com/mikeferguson/stm32/blob/9e27786de4e6df74dc735f3c5667fbe4874e59c5/libraries/STM32F4xx_StdPeriph_Driver/inc/stm32f4xx_adc.h

27. stm32/libraries/STM32F4xx_StdPeriph_Driver/inc/stm32f4xx_adc.h